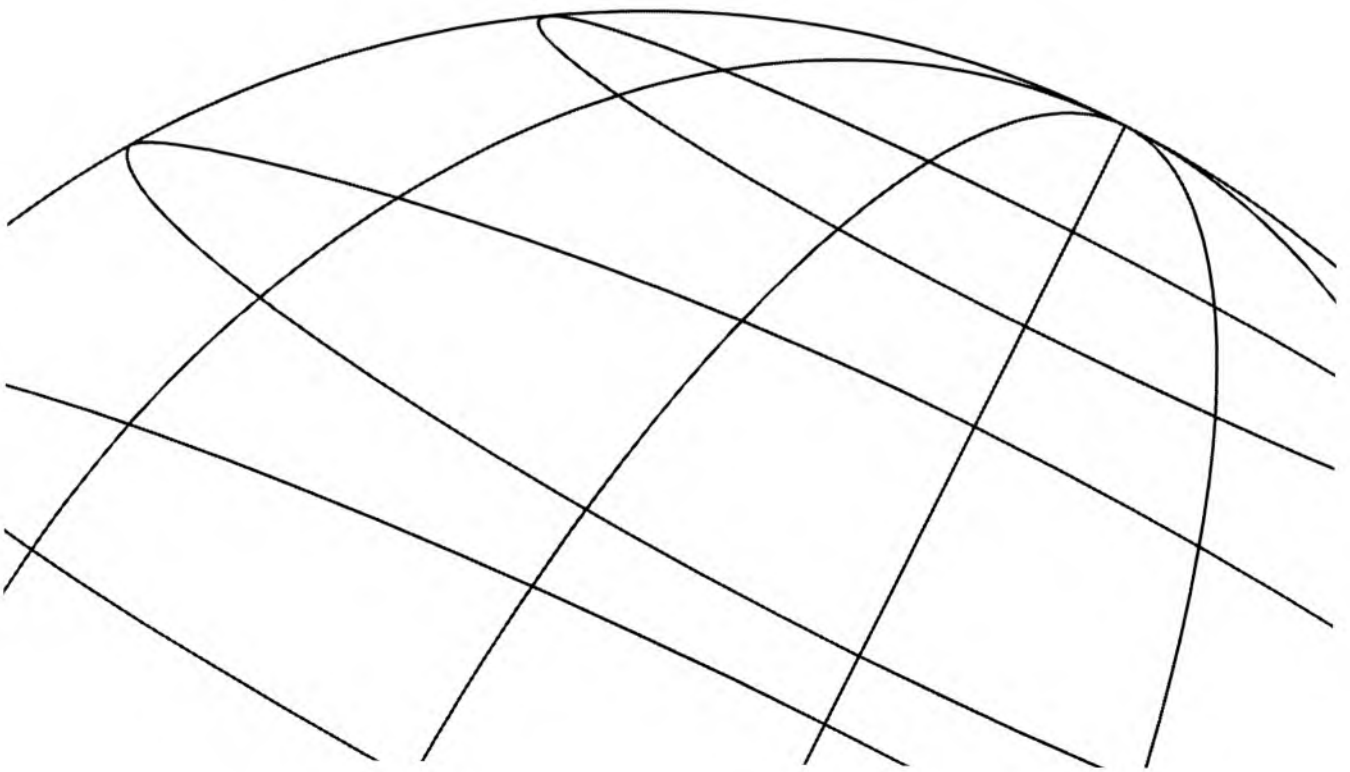
OMNIDEX

Business
Solutions for
the 90's

OMNIDEX ImagePlus SDK API Guide for HP MPE/iX Version 3.4



Dynamic Information Systems Corporation, Boulder 80301

Version 3.4
Fourth edition: June 1996

DISC

Dynamic Information Systems Corporation
5733 Central Avenue
Boulder, Colorado 80301
(303) 444-4000

DISC Europe
25-29 High Street Leatherhead
Surrey, England KT228AB
+44 1372-362777

PRINTED ON RECYCLED PAPER

Table of Contents

Using this Guide ..... **ix**
Overview ix
Conventions ix
How this Guide is Organized x
Technical support x

Acknowledgments ..... **xi**

Chapter 1: Introduction

Welcome to OMNIDEX version 3 ..... **1-2**
What are keyword keys? 1-2
What are sorted keys? 1-5

Why Use OMNIDEX? **1-7**
Speed of access to data 1-7
Retrieval flexibility 1-8
Ease of installation and maintenance 1-9
Faster application development 1-9
Reliability 1-10

Developing OMNIDEX Programs **1-11**
Using OMNIDEX intrinsics 1-12
Reducing your programming effort 1-13

Components of OMNIDEX **1-14**
The OMNIDEX utilities 1-14
The OmniUtil installation and indexing utility 1-14
The DataView query and support program 1-15

Intrinsic libraries 1-16
The OMNIDEX intrinsics library 1-16
The OMNIDEX Call Conversion library 1-17
HPPA Switch Stub intrinsic library 1-17
Why Wait any Longer for Your Data? 1-17

Chapter 2: Intrinsics

General Intrinsic Information 2-2
The OMNIDEX Intrinsic Interface 2-2
Keyword access intrinsics 2-3
Sorted access intrinsics 2-3
General intrinsics 2-4
OMNIDEX intrinsic parameters 2-4
Error handling 2-6
The TPI Interface 2-7
OMNIDEX retrievals through TurboIMAGE 2-8
Automatic updating of indexes 2-9
Error handling 2-9
Choosing a retrieval interface 2-10

Sorted Access and General Intrinsics 2-11
Mode options 2-11
Normal mode 2-11
IMAGE-only mode 2-12
Index-only mode 2-12
Sorted sequential access 2-13
DBICLOSE 2-16
Parameters 2-16
Discussion 2-17
DBIDELETE 2-18
Parameters 2-18
Discussion 2-19
OMNIDEX condition word values 2-19
DBIERROR 2-20
Parameters 2-20
Discussion 2-20
DBIEXPLAIN 2-22
Parameters 2-22

Discussion 2-22
DBIFIND 2-23
Parameters 2-23
Discussion 2-25
OMNIDEX condition word values 2-27
DBIGET 2-28
Parameters 2-28
Discussion 2-30
OMNIDEX condition word values 2-32
DBIINFO 2-34
Parameters 2-34
OMNIDEX condition word values 2-41
DBILOCK 2-42
Parameters 2-42
Discussion 2-43
DBIOPEN 2-44
Parameters 2-44
OMNIDEX condition word values 2-45
DBIPUT 2-46
Parameters 2-46
Discussion 2-47
OMNIDEX condition word values 2-48
DBIUNLOCK 2-49
Parameters 2-49
DBIUPDATE 2-50
Parameters 2-50
Discussion 2-51
OMNIDEX condition word values 2-51

Keyword Access Intrinsics 2-52

Keyword retrieval 2-52
ODXFIND 2-54
Parameters 2-54
Discussion 2-57
OMNIDEX condition word values 2-69
ODXGET 2-71
Parameters 2-71
Discussion 2-73
OMNIDEX condition word values 2-77

ODXGETWORD 2-78
Parameters 2-78
Discussion 2-79
OMNIDEX condition word values 2-79
ODXINFO 2-80
Parameters 2-80
OMNIDEX condition word values 2-85
ODXPRINT 2-86
Parameters 2-86
OMNIDEX condition word values 2-87
ODXTRANSFER 2-88
Parameters 2-88
Discussion 2-89
OMNIDEX condition word values 2-91
ODXVIEW 2-92
Parameters 2-92
Discussion 2-93
OMNIDEX condition word values 2-94

Standard Interface to Third Party Indexing 2-95

Calling TurboIMAGE intrinsics 2-95
How the interface works 2-95
OMNIDEX searches 2-96
About TurboIMAGE intrinsics 2-98
DBCONTROL 2-99
Parameters 2-99
Discussion 2-100
DBFIND 2-101
Parameters 2-101
Discussion 2-106
Calling errors and exceptional conditions 2-116
DBGET 2-117
Parameters 2-117
Discussion 2-119
Calling errors and exceptional conditions 2-122
DBINFO 2-123
Parameters 2-123
Calling errors and exceptional conditions 2-128

Chapter 3: Programming

Writing Programs **3-2**

Introduction 3-2

The OMNIDEX Intrinsic Interface 3-2

Opening databases 3-2

Updating data 3-3

Error handling 3-4

Active Error Handling 3-5

Passive Error Handling 3-9

OMNIDEX keyword retrieval 3-13

Sorted access 3-22

Standard Interface to Third Party Indexing 3-27

Opening databases 3-27

Updating data 3-28

Error handling 3-28

OMNIDEX keyword retrieval 3-29

Sorted access 3-31

Linking and Testing Programs **3-33**

Introduction 3-33

Linking and testing native mode programs 3-33

The OMNIDEX Call Conversion Library 3-34

Testing compatibility mode applications 3-36

Program capabilities 3-38

Interfacing with 3GLs **3-40**

COBOL program examples (OMNIDEX Intrinsic Interface) 3-40

DBIFIND 3-41

DBIGET 3-43

ODXFIND 3-44

ODXGET 3-45

ODXGETWORD 3-46

ODXTRANSFER 3-47

COBOL program examples (TPI Interface) 3-49

A sorted retrieval 3-49

A keyword search and retrieval 3-54

Interfacing with 4GLs **3-60**

Glossary Glossary-1

Index Index-1

Using this Guide

Overview

This preface explains the document conventions used in the *OMNIDEX API Guide* and gives you an overview of each chapter in the book.

Conventions

This guide uses the following conventions:

USER INPUT — Commands that you enter LOOK LIKE THIS. For example:

```
RUN OMNIUTIL.PUB.DISC
```

After you type the command, always press **[return]** to process that command.

`Screen text` — Prompts and messages **`look like this`**. Prompts require you to enter some information; messages are for informational purposes only. For example:

```
Enter the database name:

The database name is invalid.
```

variables — Information that you must supply, including parameters, *LOOKS LIKE THIS*. For example:

[XL.*group.account*]

Here, *group* and *account* could be any group and account.

[keys] When you are told to press a key on the keyboard, the key name is surrounded by square brackets and **[looks like this]**. For example:

Press **[return]**

Press **[ctrl]**-Y

New terms Whenever a new term is introduced, it *looks like this.*

How this Guide is Organized

- This *OMNIDEX API Guide* is organized as follows:
- This preface tells you the document conventions used in the guide and how this document is organized.
- Chapter 1, "Introduction" explains the benefits and features of OMNIDEX.
- Chapter 2, "Intrinsics" tells you how to install OMNIDEX keys, load the indexes, and test your installation.
- Chapter 3, "Programming" tells you how to call the OMNIDEX and TPI intrinsics to find records, and update OMNIDEX keys.

Technical support

If you have questions about OMNIDEX that are not addressed in this manual, you can get technical support by calling one of these numbers:

	USA and the Americas	UK and Europe
Voice:	(303) 444-6610 (production) (303) 444-4000 (trial)	+(44) 1372-362777
Fax:	(303) 444-0208	+(44) 1372-386418
Email:	support@disc.com	support@disceurope.co.uk
Address:	5733 Central Ave. Boulder, CO 80301	25-29 High St. Leatherhead Surrey, England KT228AB
Hours	7:30 AM - 6:00 PM MST	9:00 AM - 5:30 PM UK time

Acknowledgments

Dynamic Information Systems Corporation would like to acknowledge the following products, which are mentioned in this manual, and their distributors who are listed below alphabetically:

- The following are registered trademarks of Cognos Corporation: PowerHouse, QDD, QTP, QUICK and QUIZ.
- DATA Express is a registered trademark of IMACS Systems Corp.
- The following are registered trademarks of Hewlett-Packard Co.: BASIC/3000, COBOL II/3000, Compatibility Mode, DBChange, Dictionary 3000, Editor, FORTRAN/3000, HP, HP 3000, HP FORTRAN/77, HPPA, HP Word, IMAGE, MPE V, MPE XL, Native Mode, PASCAL/3000, Query, Rapid, RPG, SORT, SPL, TRANSACT/3000 and TurboIMAGE.
- Insight is a registered trademark of Unison Software Inc.
- PROTOS is a registered trademark of PROTOS Software Corporation.
- The following are registered trademarks of Robelle Consulting Ltd.: SUPRTOOL and Qedit.
- Speedware is a registered trademark of Speedware Corp.
- Synergist is a registered trademark of Gateway Systems Corp.
- Visimage is a registered trademark of Ares Corporation.

Chapter 1: Introduction

1

This chapter provides a general overview of OMNIDEX, its many powerful search capabilities and some related software products. It is divided into four sections.

- "Welcome to OMNIDEX version 3", on page 1-2, provides a general introduction to OMNIDEX's access capabilities.
- "Why Use OMNIDEX?", on page 1-7, describes a few of the benefits that OMNIDEX provides to you and your users.
- "Developing OMNIDEX Programs", on page 1-11, describes the three easy steps to using OMNIDEX.
- "Components of OMNIDEX", on page 1-14, describes the programs, utilities and procedure libraries that OMNIDEX comprises.

Welcome to OMNIDEX version 3

Welcome to OMNIDEX version 3, the powerful enhancement to TurboIMAGE. OMNIDEX uses indexes for extremely fast and flexible information retrievals. Although these indexes are transparent to your non-OMNIDEX applications, they are protected by TurboIMAGE's security and data recovery features. You can index virtually any number and type of fields in an TurboIMAGE database, including several fields in a data set. You also can design composite keys from parts of TurboIMAGE fields. OMNIDEX provides two types of keys to speed access to your data: keyword keys and sorted keys. Each of these two types of keys uses a different type of index structure to provide a different set of access capabilities. Keyword keys are discussed next. Sorted keys are discussed later.

What are keyword keys?

Keyword keys support keyword searches to find records. When you key a field for keyword access, individual data values in that field (delimited by spaces, commas and other special characters) are parsed and indexed as keywords. When you use words as arguments in a search on a keyword key, OMNIDEX searches the indexes to instantly locate all records that contain those words. This enables you to instantly locate information using words or values that may be contained anywhere in a keyed field.

Keyword keys give you the flexibility to:

- retrieve master records without knowing the search item value.
- retrieve detail records directly, without chained reads
- retrieve records by criteria in several fields simultaneously
- search on fields across sets to qualify a master and its related detail records
- search across sets in different databases

Keyword keys also support the following search operations:

- range retrievals
- relational retrievals (like greater than and less than)
- Boolean searches (AND, NOT, OR)
- generic (partial keyword) searches
- wildcard (#, ?, @) searches

Searching with keyword keys

Three CUSTOMERS master records are shown below in Figure 1-1. The underlined fields (COMPANY, CONTACT, STATE and COMMENTS) are keyword keys and the CUSTOMER-NO field is the TurboIMAGE search item (SI).

CUSTOMER-NO: 1	CUSTOMER-NO: 2	CUSTOMER-NO: 3
COMPANY: Dynamic Information Systems	COMPANY: Information Xpress	COMPANY: Dynamic Data Storage
CONTACT: Dave Smith	CONTACT: Dave Jones	CONTACT: Joan Smith
ADDRESS: 5733 Central Ave	ADDRESS: 2001 Hitek Blvd.	ADDRESS: 931 Hennepin Ave.
CITY: Boulder	CITY: Broomfield	CITY: Minneapolis
STATE: CO	STATE: CO	STATE: MN
ZIP-CODE: 80301	ZIP-CODE: 80020	ZIP-CODE: 55455
COMMENTS: aka DISC, software development company. DBMGR Omnidex, OmniView, OmniQuest OmniWindow. Sold Internationally.	COMMENTS: software consulting firm. Designs Omnidex application programs for manufacturing inventory and distribution.	COMMENTS: aka DDS. Hardware manufacturer of disk drives.

Figure 1-1: Three OMNIDEX master records

To quickly retrieve any master record with TurboIMAGE access alone, you need to know its SI (CUSTOMER-NO) value. You can retrieve master records based on the contents of a data field (like COMPANY), but that involves a serial read, which can take a while.

With OMNIDEX installed, you can search on any keyword keys to find any master record instantly. If, for example, you wanted to find a CUSTOMERS record that has "information" somewhere in the COMPANY field you could search the OMNIDEX key installed on COMPANY using the keyword argument INFORMATION.

This is called a keyword retrieval. In an online application, the search might look like this:

```
Enter a company name: INFORMATION

Records qualified: 2        List? Y

Customer number             Company
1                           Dynamic Information Systems
2                           Information Xpress
```

Records 1 and 2 qualify because they both have the word "information" somewhere in their COMPANY field. It doesn't matter where a keyword occurs in a key field. OMNIDEX keyword searches are not position sensitive.

You can install keyword keys on free form text fields, like the COMMENTS field in the sample records in Figure 1-1. The next search uses a combination of keywords to search against the OMNIDEX key installed on COMMENTS. The Boolean AND operator tells OMNIDEX to qualify records with the words "software" and "development" in the key field being searched (COMMENTS).

```
Comments: SOFTWARE AND development

Records qualified: 1        List? Y

Customer number             Company
1                           Dynamic Information Systems
```

The search qualified the only record with both "software" and "development" in the COMMENTS key field. Notice that the arguments were entered in upper and lower case. OMNIDEX keyword searches are not case sensitive by default, but can be if you want.

You can also perform keyword searches across fields. For example, if you wanted to search for records that contain "dynamic" in their COMPANY field, and "MN" in their state field, OMNIDEX would let you do so. In the prompting program, the search might look like this:

```
Enter a company name: DYNAMIC

Records qualified: 2        List? [return]

Enter the state: MN
```

```
Records qualified: 1        List? Y

Customer number             Company
3                           Dynamic Data Storage
```

Of the two records with "dynamic" in the COMPANY field, only one record (record number 3) has "MN" in its state field. Notice how OMNIDEX returns a qualifying count after each keyword search. Sometimes, this count is all the information you need.

Keyword keys are discussed in greater detail in the *OMNIDEX ImagePlus SDK Administrator's Guide.*

What are sorted keys?

IMSAM (**Im**age **S**equential **A**ccess **M**ethod) sorted keys let you retrieve records in sorted order (alphabetically or numerically) by key value. You can retrieve the records in either ascending or descending order. This is useful for a variety of applications, from generating mailing labels sorted by zip-code, to listing transactions sorted by department and date.

Indexes reduce the data redundancy usually associated with maintaining external files of sorted records, and the system overhead of maintaining pointers for sort paths. Indexes also make OMNIDEX sorted keys faster and more efficient than other means of sorting records.

Sorted keys support the following search operations:

- range retrievals
- relational retrievals (like greater than and less than)
- generic (partial key) searches
- wildcard (#, ?, @) searches (through the TPI Interface)

Searching with sorted keys

Sorted searches differ from keyword searches in that they are position sensitive. You must supply the first few (most significant) bytes of the desired key value to find records using a sorted key. If you installed a sorted key on a ZIP-CODE field, the values for that field would be stored in sorted order in the index created for that key with a pointer (CUSTOMER-NO) from each key value to its record.

Figure 1-2, below, is a flattened, partial representation of the CUSTOMERS master set and the index created for the sorted key installed on ZIP-CODE (underlined).

CUSTOMERS master data set

CUSTOMER-NO	COMPANY	.... ZIP-CODE ....
1	Dynamic Information Systems	80301
2	Information Xpress	80020
3	Dynamic Data Storage	55455
4	Tonawanda Tool and Die	14120
5	ACME Roadrunner Extermination	87109
6	Beads Unlimited	70177
7	Conglomerated Amalgams	13815
8	Flotsam Marine Salvage	21030
9	Theme-Park USA	92806
10	Five Big and Tall Guys	13815
11	McDonnels Haggisburgers	32306
12	Bates Motels of America	68504

Index set for ZIP-CODE key

Key value
13815-7
13815-10
14120-4
21030-8
32306-11
55455-3
68504-12
70177-6
80020-2
80301-1
87109-5
92806-9

Figure 1-2: Storing sorted key values

When you supply an argument in a search on a sorted key, the intrinsics compare your argument, byte for byte, against the key values stored in the index. The intrinsics then return records sorted in key order (by ZIP-CODE) to your application program.

Sorted keys also support partial key (generic) searches. These are especially useful when searching on hierarchical code fields. For example, if you supply a partial argument, like 80 against a ZIP-CODE key, you would find records whose ZIP-CODEs begin with 80.

A report of the qualifying records might look like this:

```
Customer
Number          Company                      Zip Code

2               Information Xpress           80020
1               Dynamic Information Systems  80301
```

These records represent companies in the Denver, Colorado postal zone. Imagine how useful it would be if you installed sorted keys on your larger, more complex hierarchical fields.

Sorted keys and their uses are discussed in greater detail in the *OMNIDEX ImagePlus SDK Administrator's Guide.*

Why Use OMNIDEX?

Three thousand sites world-wide use OMNIDEX. Some of the reasons are:

- speed of access to data
- retrieval flexibility
- ease of Installation and Maintenance
- faster Application Development
- reliability

Each of these reasons are discussed next.

Speed of access to data

Probably the biggest reason to use OMNIDEX is the speed with which it can locate records. With TurboIMAGE you don't have to worry about speed as long **as you always search on key fields using full key values**. If you search on a field that is not an TurboIMAGE key you have to wait while your application program serially reads every record and vacant address in the data file. If you have half a million master records in your data file, the search would take 55 minutes[1], whether it qualified five records or five thousand records.

If you were looking for a specific topic in a book, you wouldn't read the book from cover to cover until you found the topic; you would look for the topic in the index and go directly to the pages that contained information about that topic. Because OMNIDEX uses indexes to locate data, it eliminates the need for serial reads. OMNIDEX's indexes use record identifiers like the index of a book uses page numbers. When you search on a keyword or sorted key, OMNIDEX searches the indexes for the requested key values. Then, OMNIDEX returns the record identifiers to your application program, which uses them to get the actual records.

1. Assume that the blocking factor is 6 and that the disc access time is 30 IOs per second.

OMNIDEX can qualify between 40,000 and 225,000 records per second, depending on CPU class. It can reduce search times from hours to seconds in many cases. This savings in time and system overhead means that you can get your data right away, in real time, online. It also means increased throughput, and a more efficient use of system resources.

Retrieval flexibility

OMNIDEX lets you key any number of fields in any set, including TurboIMAGE masters for instant keyword retrieval and. You can install these keys on an existing database in minutes, without restructuring it. Almost any field can be keyed for OMNIDEX access, which allows users far greater flexibility when searching for records.

OMNIDEX lets you index and retrieve by individual words or values (keywords) in a field. This means you can find a CUSTOMERS record, for example, without knowing an TurboIMAGE SI value. All you need is some part of the company name, the contact's first or last name, even the city, or the name of a product that the customer may have purchased. This capability is invaluable to customer service representatives, as well as other users.

Sorted keys let you retrieve records in order, sorted by key values. If you create a composite key that combines several fields, you can sort records by several fields. This enables you to generate reports, like mailing labels, or general ledgers, that are sorted for easy reference.

As you learn more about OMNIDEX's many features and discover new uses for them, you can easily add them to your existing installation. Your users will like OMNIDEX's many access features, which let them ask for data in their own language instead of TurboIMAGE key values. OMNIDEX also supports native language extended character sets (like Roman8 and Turkish8).

In addition, OMNIDEX intrinsics can be called through many programming languages. Therefore, you can use OMNIDEX through a screen formatting language, like QUICK, to provide friendly screens and menus for your end users.

Ease of installation and maintenance

OMNIDEX's OmniUtil program lets you install OMNIDEX keys on your database, or change how they are installed, in a matter of minutes. OmniUtil is very easy to use, and contains online help at each prompt. When you are finished specifying keys for installation, OmniUtil creates the required indexes automatically at your convenience.

The data integrity of the OMNIDEX indexes is maintained automatically. The OMNIDEX intrinsics update the indexes to reflect any data that is added, updated, or deleted in your TurboIMAGE database. Your current TurboIMAGE based data entry programs can be run, unchanged, through OMNIDEX's Call Conversion library or through the Standard Interface to Third Party Indexing (the TPI Interface) to automatically update the indexes.

If your update programs call the OMNIDEX intrinsics, you can continue to use them with the version 3 intrinsics.

The most work to installing OMNIDEX is in deciding what keys to install on your database. The *OMNIDEX ImagePlus SDK Administrator's Guide* contains all the information you need to decide how you want to install OMNIDEX.

Faster application development

OMNIDEX version 3 gives you the choice to use the traditional OMNIDEX Intrinsic Interface, or the Standard Interface to Third Party Indexing (TPI).

The OMNIDEX Intrinsic Interface

OMNIDEX intrinsics are similar to TurboIMAGE intrinsics, so they are easy to learn and incorporate into your programs. They can be called through many software packages and programming languages.

OMNIDEX intrinsics are portable. When OMNIDEX is ported to other database management systems, and other operating systems, you can use the applications you develop using the OMNIDEX Intrinsic Interface with little or no change. So if you plan to migrate to a different database management system or a different operating system, you may want to develop applications using the OMNIDEX Intrinsic Interface.

Because the OMNIDEX intrinsics are capable of Boolean and relational logic, programmers benefit by being able to eliminate a substantial amount of IF, THEN, ELSE logic from their program code. OMNIDEX offers a wide variety of indexing options, which can eliminate much of the upshifting, byte addressing, and other data manipulation from your application programs.

Dynamic Information Systems Corporation (DISC) can provide specific information, and in many cases, procedure source code examples, for your programming language. To add OMNIDEX retrieval capability to existing online programs without programming changes, you might consider OmniWindow. OmniQuest, available from DISC, lets you perform OMNIDEX retrievals from report writers and transaction processors like QUIZ, QTP and Query, or from your batch programs. Complete application interfaces to OMNIDEX are also available for ASK/MANMAN, Multiview, MCBA, Jobscope, and Gerber-Alley.

The TPI Interface

If you use TurboIMAGE version C.04.03 or later, you can perform OMNIDEX retrievals and updates by calling TurboIMAGE intrinsics at the system level. Because this interface to OMNIDEX is integrated at the system level, it optimally manages concurrent operations, and provides automatic recovery of TurboIMAGE and OMNIDEX data if a system failure occurs.

Reliability

OMNIDEX has been in use since 1985 and is installed on over three thousand sites world wide. Many third party vendors have included OMNIDEX into their information management software products. DISC's product development staff continually refines OMNIDEX to make it easier to use and better suited to individual applications.

OMNIDEX owes much of its reliability to its compatibility with TurboIMAGE. The OMNIDEX indexes are protected by the umbrella of TurboIMAGE security features, which includes transaction logging and dynamic rollback recovery. They are also entirely compatible with your current information management procedures.

Developing OMNIDEX Programs

Developing applications for use against an OMNIDEX-enhanced database requires the few steps discussed below. The first of these steps insures that your current applications can still be run against your OMNIDEX-enhanced databases. Subsequent steps will acquaint you with the things you can do through your programs, and how to do them.

Step 1: Copy the OMNIDEX procedure libraries

OMNIDEX comes with several procedure libraries. These include:

- the OMNIDEX intrinsic library in XLOMNIDX.PUB.DISC
- the Call Conversion library in XL.PUB.DISC
- switch stubs in SL.PUB.DISC, for converting compatibility mode calls to native mode calls

You must copy the OMNIDEX intrinsic library (XLOMNIDX) to PUB.SYS. If you used the SETUP program when you installed the software tape, this was done automatically. Then, when your programs reference the appropriate procedure library, your applications can use the power of OMNIDEX. For information on installing and referencing the OMNIDEX procedure libraries, see Appendix C of the *OMNIDEX ImagePlus SDK Administrator's Guide.*

Step 2: Use DataView to access your OMNIDEX-enhanced database

DataView, DISC's powerful database access tool, uses OMNIDEX intrinsics to perform searches. It is a good tool to use when developing retrieval applications because it gives you ideas as to what your OMNIDEX-based programs can do. You can also use it to test retrievals that your programs perform. For more information about DataView, see the "Utilities" chapter of the *OMNIDEX ImagePlus SDK Administrator's Guide.*

Step 3: Develop your own applications

This manual provides general information about developing programs to access OMNIDEX-enhanced databases. There are also source code files, written in a variety of programming languages, in the DEMO group of your DISC account. They can be compiled and run, or copied and modified, to provide an introduction to OMNIDEX programming.

In addition to this manual, several specialized manuals are available for the following topics:

- The *OMNIDEX Language Sampler* provides information and examples of OMNIDEX programs written in FORTRAN, BASIC, PASCAL, and RPG.
- The *OMNIDEX PROTOS Interface Guide* discusses how to write OMNIDEX-based applications through PROTOS.
- The *OMNIDEX QUICK Interface Guide* discusses how to create OMNIDEX-based QUICK applications.
- The *OMNIDEX RAPID Interface Guide* discusses how to create OMNIDEX-based RAPID applications.

Using OMNIDEX intrinsics

Developing OMNIDEX-based applications is not difficult when you consider that OMNIDEX is an enhancement to your DBMS. As such, its intrinsics include OMNIDEX versions of most of the TurboIMAGE intrinsics. For example, the parameters used by TurboIMAGE's DBPUT intrinsic are the same as those used by OMNIDEX's DBIPUT.

The major differences between TurboIMAGE and OMNIDEX intrinsic calls are:

- There are more OMNIDEX intrinsics than TurboIMAGE intrinsics. Besides including enhanced versions of all of the TurboIMAGE intrinsics, the OMNIDEX intrinsics provide a variety of additional features.
- There are more OMNIDEX mode values than TurboIMAGE mode values. These additional mode values are passed through your programs to enable OMNIDEX's many features.

While there is more to learn when developing OMNIDEX applications, the syntax for calling OMNIDEX intrinsics does not require a major change in your thinking. What you can do with the OMNIDEX intrinsics does require a leap of your imagination.

Reducing your programming effort

The many features of OMNIDEX can actually simplify your programs. These features are incorporated at the intrinsic level. This can eliminate much of the coding effort involved in tailoring data retrieval programs to meet user's needs. Many of the search operations that must be coded through IF..ELSE logic can be replaced with the Boolean, relational, and range operations supported through OMNIDEX. Similarly, ODXFIND modes 3 and 5 (enhanced parsing) can convert date search arguments into proprietary date formats, such as ASKDATE and PHDATE formats.

Direct access to masters and details through keyword and sorted keys can eliminate many of the chained reads that were necessary through TurboIMAGE. Because keys can be created from parts and combinations of fields, the right keys can eliminate having to programmatically match arguments against certain bytes of a field.

Because your TurboIMAGE update programs can automatically update indexes, you don't have to worry about developing new update programs. You can, however, modify them to call the OMNIDEX update intrinsics, or to perform specialized updates to maximize performance.

Components of OMNIDEX

The OMNIDEX Information Management System consists of the following components:

- OMNIDEX utilities
- Intrinsic libraries

These components, which are listed below according to function, support the complete implementation and maintenance of OMNIDEX on your databases.

The OMNIDEX utilities

OMNIDEX includes two utilities:

- OmniUtil, used to install and maintain OMNIDEX keys
- DataView, used to find records using OMNIDEX keys

Both of these utilities are easy to use. You need only highlight a menu selection and press **[return]** to install keys, reload indexes, or search for records. Both utilities are discussed next.

The OmniUtil installation and indexing utility

OmniUtil is a program that combines all the OMNIDEX installation, indexing, configuration and maintenance operations into one menu-driven interface. You can use OmniUtil to install and maintain OMNIDEX.

To run OmniUtil, enter the following command at the system prompt:

```
RUN OMNIUTIL.PUB.DISC
```

The main OmniUtil menu screen appears, as shown in Figure 1-3.

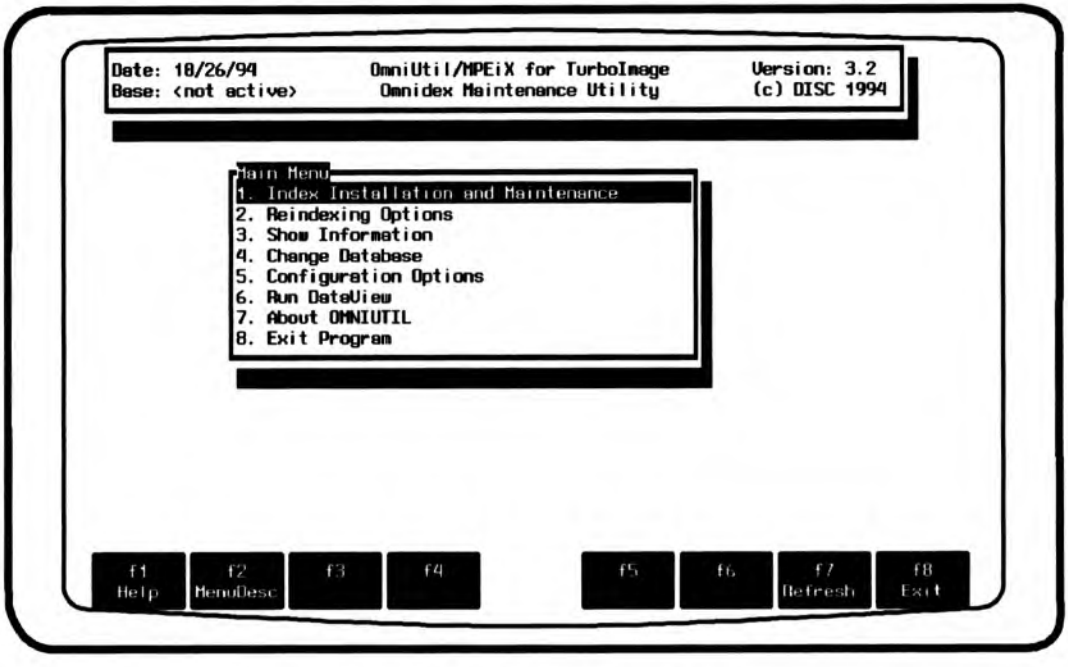


Figure 1-3: The OmniUtil Main Menu

See the "OMNIDEX Installation" chapter in the *OMNIDEX ImagePlus SDK Administrator's Guide* for detailed information about OmniUtil's menus and its use in installation and index maintenance.

The DataView query and support program

DataView is a menu driven database query tool designed for OMNIDEX databases. DataView lets you find records using OMNIDEX keyword and sorted keys.

Almost anyone can learn to use DataView. It lets you test the OMNIDEX keys you installed on your database while you develop more specialized applications.

Intrinsic libraries

Intrinsics are callable procedures used in your application programs for data retrieval, updating records and other functions related to your TurboIMAGE database. Several intrinsic libraries are included in OMNIDEX:

- The OMNIDEX intrinsics
- The OMNIDEX Call Conversion library
- The HPPA switch stub library

Each of these libraries are discussed briefly below, and in more detail in the *OMNIDEX ImagePlus SDK Administrator's Guide*.

The OMNIDEX intrinsics library

The OMNIDEX intrinsics are similar to TurboIMAGE intrinsics. Unlike TurboIMAGE intrinsics, they perform OMNIDEX-related functions, like maintaining the indexes, and performing keyword and sorted retrievals.

The OMNIDEX intrinsics used to perform sorted and multiple keyword retrievals reside in the same library. Keyword and sorted keys, however, each use different index structures to provide their different retrieval capabilities.

Call OMNIDEX intrinsics through your application programs to provide keyword retrieval and sorted access capabilities, which were described earlier. OMNIDEX intrinsics also interact with TurboIMAGE intrinsics to:

- transfer retrieved information to a file
- return information about a database and the OMNIDEX keys installed on it
- handle OMNIDEX errors

When intrinsics perform general functions (such as OMNIDEX index maintenance), they are called *general intrinsics*. The OMNIDEX general intrinsics can be substituted for almost all of the TurboIMAGE intrinsics. The exceptions are DBBEGIN, DBEND and DBMEMO, which have no OMNIDEX counterparts.

OMNIDEX *sorted access* and *keyword access intrinsics* perform all of the sorted sequential and keyword retrieval operations, discussed earlier, plus any standard TurboIMAGE retrievals.

The OMNIDEX Call Conversion library

The OMNIDEX Call Conversion library intercepts calls to OMNIDEX and TurboIMAGE intrinsics, and directs them to the OMNIDEX intrinsic library. This Call Conversion library lets you run your existing update applications to automatically update the OMNIDEX indexes as you update your TurboIMAGE data. This is useful for databases that have not been enabled for TPI, but require real time updating through TurboIMAGE applications.

HPPA Switch Stub intrinsic library

A switch stub intrinsic library is provided to enable compatibility mode programs to use MPE/iX based OMNIDEX software. The library is shipped as SL.PUB.DISC. All you need to do is copy it, along with the OMNIDEX Call Conversion library, into your compatibility mode application accounts and run your applications with the appropriate LIB parameter. The routines in the Switch Stub intrinsic library convert compatibility mode calls to intrinsics into native mode calls to OMNIDEX or TurboIMAGE intrinsics, as appropriate. For more information about the Switch Stub intrinsic library, see Appendix C of the *OMNIDEX ImagePlus SDK Administrator's Guide*.

Why Wait any Longer for Your Data?

In the pages that follow, you will learn how to install all of the features of OMNIDEX on your databases. The OMNIDEX documentation provides all the information you need to implement OMNIDEX.

For those of you who might benefit from hands-on training, DISC offers intensive OMNIDEX training classes. The classes are taught by members of our technical support staff. Training involves instruction combined with extensive lab work in implementing OMNIDEX's capabilities on the student's own applications. Classes are held at our home office in Boulder, Colorado, and at other select locations throughout the United States. For schedules or more information about OMNIDEX training classes, contact your DISC sales representative at (303) 444-4000.

Chapter 2: Intrinsics

This chapter describes the intrinsics that your application programs can call to provide OMNIDEX retrieval capabilities on an OMNIDEX-enhanced database.

- "General Intrinsic Information", on page 2-2, provides an overview of the intrinsics.
- "Sorted Access and General Intrinsics", on page 2-11, provides detailed information about each OMNIDEX "DBI" intrinsic, including syntax and parameters.
- "Keyword Access Intrinsics", on page 2-52, provides detailed information about each OMNIDEX "ODX" intrinsic, including syntax and parameters.
- "Standard Interface to Third Party Indexing", on page 2-95, provides detailed information, including syntax and parameters, about the TurboIMAGE intrinsics that you can use to access OMNIDEX.

You also should read the "Programming" chapter of this manual for information about how to call the intrinsics through application programs.

General Intrinsic Information

You can use either of two sets of intrinsics to search and maintain OMNIDEX indexes. The *OMNIDEX Intrinsic Interface* includes all of the intrinsics that past users of OMNIDEX are familiar with, such as ODXFIND, ODXGET, and DBIGET. The OMNIDEX Intrinsic Interface lets you easily migrate your applications as OMNIDEX is ported to other operating systems and database management systems.

Sites that use TurboIMAGE version C.04.03 or later, or MPE/iX version 4.0 or later, can use the Standard Interface to Third Party Indexing (or *"TPI Interface"*). This interface provides OMNIDEX retrieval capabilities through the familiar TurboIMAGE DBFIND and DBGET intrinsics. Because this interface is integrated at the operating system level, it handles concurrent operations more efficiently, and provides full recovery of OMNIDEX indexes along with TurboIMAGE data.

The OMNIDEX Intrinsic Interface

OMNIDEX provides retrieval intrinsics that access the OMNIDEX keyword and sorted indexes. Additional intrinsics are provided to perform maintenance tasks like locking and updating OMNIDEX indexes. The retrieval intrinsics that access keyword keys are called *keyword access intrinsics*. The retrieval intrinsics that access sorted keys are called *sorted access intrinsics*. The remaining maintenance intrinsics are called *general intrinsics*. When you use the keyword or sorted access intrinsics to do your retrievals, you are using the OMNIDEX Retrieval Interface.

Keyword access intrinsics

The Keyword access intrinsics retrieve or process records using the OMNIDEX keyword indexes. Keyword access intrinsics are called to:

- qualify records based on any combination of keyword values contained in OMNIDEX keyed fields
- return information about keyword keys, domains and databases
- transfer qualified records or their items to files

The Keyword access intrinsics are:

- ODXFIND
- ODXGET
- ODXGETWORD
- ODXINFO
- ODXPRINT
- ODXTRANSFER
- ODXVIEW

Sorted access intrinsics

Sorted access intrinsics are called to do sorted-sequential retrievals, based on partial, or full, key values entered for a sorted key. They accomplish this by accessing the index for each sorted key. The sorted access intrinsics are:

- DBIFIND
- DBIGET

These intrinsics use the same number and types of parameters as their corresponding TurboIMAGE intrinsics (DBFIND and DBGET). Note that DBIFIND's and DBIGET's parameters are slightly different than their TurboIMAGE counterparts. The major difference is that DBIGET and DBIFIND support several additional *mode* values. However, DBIFIND and DBIGET perform standard TurboIMAGE functions if a standard TurboIMAGE mode value is used.

General intrinsics

Some OMNIDEX intrinsics are called general intrinsics. They are used to:

- open and close OMNIDEX-enhanced databases
- perform puts, deletes, and updates and automatically perform the corresponding put, update or deletion of keys or keywords in the OMNIDEX indexes
- return error messages and information about a database

The general intrinsics replace, or supplement, most of the TurboIMAGE intrinsics on a one-for-one basis. They are:

- DBICLOSE
- DBIDELETE
- DBIERROR
- DBIEXPLAIN
- DBIINFO
- DBILOCK
- DBIOPEN
- DBIPUT
- DBIUNLOCK
- DBIUPDATE

OMNIDEX intrinsic parameters

OMNIDEX intrinsics use the same number and types of parameters as their corresponding TurboIMAGE intrinsics. Most OMNIDEX intrinsics (except for DBIEXPLAIN, see below) execute identically to their TurboIMAGE counterparts if a standard TurboIMAGE mode value is used. This means that a program where TurboIMAGE intrinsics are replaced by OMNIDEX general intrinsics (with no other changes) operates identically to the TurboIMAGE version. There are some differences.

- OMNIDEX permits several additional values for the mode parameter. For example, DBIGET and DBIFIND provide relational modes that support generic (partial) specification of the argument value.

- Certain OMNIDEX intrinsics use mode options. These are integer values that are added to the mode parameter values to augment the specified mode's operation.
- DBIEXPLAIN has an additional parameter called *parm*, which enables you to configure how errors are handled.
- The DBIGET *dset* parameter can be 16 words (32 bytes) long, and can include an 8 word (16 byte) name of a sorted key.
- OMNIDEX intrinsics require a 21 word status array instead of the 10 word array required in calls to TurboIMAGE intrinsics.Words 1-10 are still used for the information returned by the TurboIMAGE call, word 11 is the OMNIDEX condition word, and words 12-21 vary according to the intrinsic being called.

Types of modes and mode options

In addition to the standard TurboIMAGE mode values, the sorted access and general intrinsics accept additional mode values when performing sorted retrievals and other functions.

For general intrinsics, the three types of mode options are described by the data they affect.

- *Normal mode* affects both the TurboIMAGE data sets and the OMNIDEX indexes, and is used in most cases.
- *IMAGE-only mode* affects only the TurboIMAGE data sets, and is used for extensive batch updates.
- *Index-only mode* affects only the OMNIDEX indexes, and is used to perform index-only mode retrievals and updates.

These mode options are discussed in "Mode options", on page 2-11.

Standard TurboIMAGE mode values for sorted access and general intrinsics are marked with an asterisk (*) throughout in the "Sorted Access and General Intrinsics" section of this chapter.

Error handling

In the OMNIDEX Intrinsic Interface, there are two ways to handle errors:

- Default, or "active", error handling returns all error condition values to word 1 of *status*. This method of error handling is best suited to OMNIDEX applications.
- Passive error handling, when enabled for an XL, returns indexing error conditions to word 12 of *status*. This method of error handling is designed for non-OMNIDEX update applications running through Call Conversion to enable automatic indexing. Passive error handling is not supported for TPI-enabled databases.

This section discusses the default, active error handling for the OMNIDEX Intrinsic Interface. For more information about either type of error handling, see "Error handling", on page 3-4 of the Programming chapter.

The condition word

The *status* array for OMNIDEX intrinsics is a 21 word array. Under default error handling, words 1 through 10 return TurboIMAGE errors, just like the status array used by TurboIMAGE intrinsics. The first word of the status array is used as the TurboIMAGE condition word, which indicates whether or not a call has executed successfully. If this word contains a nonzero value, an error was incurred by the calling intrinsic.

OMNIDEX *status* array values

Table 2-1, on the next page, lists the values that are returned in the *status* array for any given intrinsic call.

TurboIMAGE condition word 1 value	*Status* word 11 value	Indication
0	0	Successful execution; no errors and no warnings
+888	negative	Keyword access calling error
+888	positive	Keyword access exceptional condition
+999	negative	Keyword access or general calling error
+999	positive	Keyword access or general exceptional condition
other nonzero	0	Primary TurboIMAGE call error; words 1-10 contain TurboIMAGE call data
other nonzero	positive	Secondary TurboIMAGE call error; words 1-10 contain TurboIMAGE call data

Table 2-1: How status array values indicate errors

The TPI Interface

With the release of TurboIMAGE version C.04.03 and the Standard Interface to Third Party Indexing, you can perform OMNIDEX searches and updates by calling TurboIMAGE intrinsics at the system level. If you purchased TurboIMAGE with MPE/iX version 4.0 or later, you have this capability. When a database is enabled for TPI, any programs you wrote using the TurboIMAGE Retrieval Interface under version 2.09/2.10, and your existing TurboIMAGE programs, can resolve through the system XL to interface with OMNIDEX.

The TPI Interface provides the following features:

- record counts for both sorted and keyword keys
- range retrievals on sorted keys
- familiar DBFIND/DBGET programming logic for both sorted and keyword keys
- automatic updating of OMNIDEX indexes

OMNIDEX versions 2.09 and 2.10 emulate the retrieval capabilities of the Standard Interface to Third Party Indexing, but do not provide the level of integration found in TurboIMAGE version C.04.03 and OMNIDEX version 3.

OMNIDEX retrievals through TurboIMAGE

You can perform keyword searches and sorted retrievals using DBFIND and DBGET. These standard TurboIMAGE intrinsics have been enhanced with new routines, available through new mode values. They support many existing, and a few new, retrieval features including:

- ranges on sorted keys
- relational (>, >=, <, <=) searches
- Boolean operations (performed in the order NOT, AND, then OR)
- pattern matched or "wildcard" searches
- generic or partial key access

The only difference between the TurboIMAGE retrieval intrinsics and the traditional TurboIMAGE intrinsics are new mode values. However, DBFIND mode 1 is enhanced to provide sorted and keyword access. This means that you can use the retrieval features of OMNIDEX through existing applications (according to the conventions specified in Hewlett-Packard's Standard Interface to Third Party Indexing).

Programs that use the Standard Interface to Third Party Indexing can access OMNIDEX databases that are not enabled for TPI by resolving calls through the OMNIDEX Call Conversion library, as discussed in Appendix C of the *OMNIDEX ImagePlus SDK Administrator's Guide*.

Automatic updating of indexes

When you run update programs against a TPI-enabled database, additions, deletions, and updates of records are automatically reflected in the OMNIDEX indexes. TurboIMAGE intrinsics like DBOPEN, DBPUT, DBDELETE, and DBUPDATE have been modified to detect the presence of indexes, and to update them synchronously with the TurboIMAGE records in sets that contain keys. You can disable real time updating of indexes by disabling the indexes for TPI through OmniUtil. This is discussed under "Updating OMNIDEX Data" in the "Topics" chapter of the *OMNIDEX ImagePlus SDK Administrator's Guide.*

Error handling

The TurboIMAGE intrinsics use a 10-word status array, just like the TurboIMAGE intrinsics you are used to using. The major difference in error handling between the TPI Interface and normal TurboIMAGE is the error codes. As in normal TurboIMAGE, calling error codes are negative numbers, and exceptional condition codes are positive numbers. Normal TurboIMAGE error codes are unchanged. For example, error 14 is still an exceptional condition, and means that the beginning of a chain has been reached.

Error codes that pertain to OMNIDEX indexes are four digit codes that begin with the number 3. For example, error -3304 is a calling error, the same as OMNIDEX error -304, which means that the field passed in *item* is not a sorted key.

Choosing a retrieval interface

Before you begin developing OMNIDEX applications, you should decide if you want to use the OMNIDEX Intrinsic Interface or the TPI Interface. The OMNIDEX Intrinsic Interface requires the OMNIDEX intrinsics library (XLOMNIDX.PUB.SYS), and provides capabilities beyond those provided in the TPI Interface. Some of these additional capabilities include:

- keyword-only retrievals (keyword lookups)
- Soundex (phonetic) searches
- index-only mode sorted retrievals (key value only)
- transfer of fields from records qualified in keyword searches

The TPI Interface provides capabilities beyond the OMNIDEX Intrinsic Interface. These capabilities include:

- range retrievals on sorted keys
- record counts on sorted searches
- pattern matching in sorted searches
- familiar coding techniques
- automatic recovery of OMNIDEX indexes after a system failure

You should choose an interface based on the set of features you require, and the possibility of migrating to another database management system (such as SYBASE or Oracle) or operating system (such as HP-UX or OpenVMS).

Sorted Access and General Intrinsics

OMNIDEX sorted access intrinsics let you search OMNIDEX sorted (IMSAM) keys. OMNIDEX general intrinsics perform the same function as their TurboIMAGE counterparts, but they extend their actions to any OMNIDEX indexes associated with their target data object.

Mode options

One of the differences between TurboIMAGE and OMNIDEX *mode* parameter values is OMNIDEX's use of *mode options*. Mode options enable you to control what data structures are targeted by the general intrinsics through the use of three mode designations, which are discussed next:

- normal mode
- IMAGE-only mode
- index-only mode

Normal mode

Normal mode operations affect both TurboIMAGE data sets and the OMNIDEX indexes. For the DBIFIND and DBIGET sorted access intrinsics, a key is first located in an index. DBIFIND uses this key to locate a chain head and set up chained access to a detail set. DBIGET uses the key to retrieve a record from the data set.

Normal mode calls to DBIPUT must use an @ item list.

For the update intrinsics, DBIPUT, DBIDELETE and DBIUPDATE, an entry is first added to, or deleted from, the data set specified by the *dset* parameter. Then, the corresponding OMNIDEX indexes are updated automatically.

IMAGE-only mode

An IMAGE-only mode option is available for the update intrinsics DBIPUT and DBIDELETE. The IMAGE-only mode option causes these update intrinsics to affect only the TurboIMAGE data sets. The OMNIDEX indexes are not updated by intrinsics called in IMAGE-only mode.

IMAGE-only updates enable you to eliminate the overhead associated with updating OMNIDEX indexes in real time. The indexes can be updated later by an OmniUtil indexing operation.

Index-only mode

The index-only mode option is available for the retrieval intrinsics DBIFIND and DBIGET.

Index-only mode DBIGET is used to retrieve only a sorted key, not its corresponding entry. This is useful because it is much faster than normal mode. An index-only mode DBIGET need only retrieve the desired key from an index; a normal mode DBIGET also must read the corresponding record from the data set.

An index-only mode DBIGET can be used whenever only the key value is needed. Often, a database administrator designs composite sorted keys to be accessed in index-only mode. They combine fields to be searched (as the first components in the key) with fields that contain the desired data (as the last components in the key). This is useful for reporting, in that the overhead required to get TurboIMAGE records is eliminated.

If several sequential retrievals are being performed, index-only mode is much faster than normal mode. The sorted access intrinsic keeps the most recently accessed tree block in the user's stack. Because many keys reside in one physical block, a full block of keys can be retrieved with a single read to disk.

Index-only mode usually returns keys 20-100 times faster than normal mode for sequential retrieval operations. This advantage is greater for smaller keys.

Index-only mode is also used to mark a place during a sorted read. This technique is necessary when using a sorted key to find records for updating. If you update the sorted key that is used to read records in sorted order, the old sorted key value is removed from the index, and the new, updated key value is placed in the proper location in sort order. So,

the next sorted sequential read (mode 90 or 91) reads from the new key value, not from where the old key value was deleted.

To avoid this, and continue reading from where you deleted the old key value, use index-only mode to store the full internal key value before the update. Then, use this stored key value and mode 300 to redirect DBIGET to the old key's location before continuing your read.

Sorted sequential access

The DBIFIND and DBIGET sorted access intrinsics provide keyed sequential access and partial-key (generic) retrieval capabilities through sorted keys:

- DBIFIND locates detail chain heads using a sorted key installed on the search item of a master.
- DBIGET retrieves records or key values using all other sorted keys.

The initial record is found by matching an argument value (passed through the *argument* parameter) against the key values stored in the key's index. When a key value that matches the argument value is found, the record that it corresponds to is retrieved. Subsequent records can be returned in sorted order by calling DBIGET to read up or down through the index, depending on the *mode* value.

DBIFIND and DBIGET let you find or retrieve records without fully specifying the key value passed through the *argument* parameter. This is called a *generic retrieval*. Generic (partial) key values are specified starting with the leftmost character. The *mode* parameter specifies the length of the partial key value in bytes or words. It also specifies a relation between the *argument* value supplied and the key value to be retrieved.

The DBIFIND and DBIGET relational operations and modes, where *nn* represents the partial *argument*'s length, are:

=	equal to	mode 1*nn*
>	greater than	mode 2*nn*
>=	greater than or equal to	mode 3*nn*
<	less than	mode 4*nn*
<=	less than or equal to	mode 5*nn*

For example, a CUSTOMERS master data set might have a sorted key on the 32-byte COMPANY field. The COMPANY key values, and pointers to the records that contain them, are stored in an associated index. The key values and their records are accessible through DBIGET.

To retrieve a CUSTOMERS master entry, call DBIGET, specify from 1 to 32 characters of a COMPANY name (in *argument*, starting with the leftmost character). Then, specify the desired relationship (like > or <) between the *argument* value and the key sought using one of the relational mode values shown above. Substitute the length of the argument, if it is generic, for *nn* in *mode*.

Note that the scope of a relational retrieval is determined by the length of the argument specified for the key value. The number of leading characters that are matched against indexed key values determines which record is initially selected.

For example, an equal to (=) relational search on a COMPANY key using the generic argument A (mode -101) would retrieve the first record that begins with an A (like AARDVARK). Subsequent DBIGET reads (via the sequential modes listed below) would return records with key values like ABEL, and ACME. The argument AC qualifies the records for ACE and ACME, but not ABC, because the two characters do not match the partially specified argument value AC.

The more characters that are specified, the more precise the retrieval criteria, and the more selective the retrieval.

The key value returned by a sorted retrieval depends on the specified relation and argument. For the relational operations =, > and >=, the first value returned is the first matching key value in ascending sequence, or the lowest key value that qualifies. For the relational operations < and <=, the first value returned is the first key value in descending key sequence, or the highest key value that qualifies.

For example, an index for a sorted key might contain several key values starting with L, ranging from LABEL to LUCKY. A partial key = or >= operation using the argument value L would return LABEL, while a <= operation would return LUCKY.

Similarly, a > operation using the argument value L would return the lowest key starting with M, and a < operation using the argument value L would return the highest key starting with K. Some possible results of relational retrievals are listed below.

Operation and Value	First Key Value Found
<L	KUDOS
=L	LABEL
>=L	LABEL
<=L	LUCKY
>L	MAN

After you locate a key value using a DBIFIND or DBIGET relational mode value, use one of DBIGET's sequential modes to read through the indexed (sorted) key values. The DBIGET sequential reads and their modes are:

90	ascending read	next higher value
91	descending read	next lower value
92	reread	same value

For more information about sorted access, see the DBIFIND and DBIGET headings in this chapter, and "Writing Programs", on page 3-2 of the "Programming" chapter.

The rest of this section discusses each of the OMNIDEX sorted access and general intrinsics. The intrinsics sections are arranged alphabetically. Each includes information about the intrinsic's parameters, a discussion of what it does, and a list of possible error numbers and messages.

DBICLOSE

DBICLOSE (*base, dset, mode, status*)

DBICLOSE terminates an access path to a database, or closes or rewinds a data set. It also recovers any resources used by OMNIDEX to access the database.

Parameters

base is the name of the integer array used as the *base* parameter when opening the database. The first element of the array must contain the base ID returned by DBIOPEN. (See page 2-44 for more information about the base ID.)

dset is the name of an array that contains the left-justified name, or the 16-bit number, of the data set being accessed. The data set name may be up to 16 characters long or, if shorter, terminated by a semicolon (;) or a space (for example, CUSTOMERS; or ORDER-LINES).

mode is a 16-bit word integer from 1 to 3. The values with asterisks (*) correspond directly to the TurboIMAGE DBCLOSE *mode* values as follows:

- 1 * terminates access to the database via a path. The path is identified by the first word of the *base* parameter, which was assigned by TurboIMAGE during DBIOPEN.
- 2 * closes the data set specified in the *dset* parameter.
- 3 * rewinds (re-initializes) the data set specified in the *dset* parameter.

status is the name of an array of 21 16-bit words used to return information about the success of a call. See Table 2-1, on page 2-7, for a list of error condition codes.

Discussion

DBIOPEN must be used to open an OMNIDEX database when using the OMNIDEX Intrinsic Interface. Multiple databases may be opened, and each database may be opened multiple times. The number of databases you can have open at any one time is roughly 10, depending on the size and complexity of the databases and the number of excluded words.

DBICLOSE modes 2 and 3 are identical to the corresponding DBCLOSE mode values.

DBICLOSE mode 1 first issues a DBCLOSE mode 1 to terminate access to the database via the path specified by the base ID (the first word in the *base* parameter) and releases resources that are used to control access against the database. If multiple databases are accessed serially instead of concurrently, you should close one database before opening the next to conserve memory resources.

DBIDELETE

DBIDELETE (*base, dset, mode, status*)

DBIDELETE deletes the current entry from the master or detail set referenced in *dset* and updates any affected indexes.

Parameters

base is the name of the integer array used as the *base* parameter when opening the database. The first element of the array must contain the base ID returned by DBIOPEN. (See page 2-44 for more information about the base ID.)

dset is the name of an array that contains the left-justified name, or the 16-bit number, of the data set being accessed. The data set name may be up to 16 characters long or, if shorter, terminated by a semicolon (;) or a space (for example, CUSTOMERS; or ORDER-LINES).

mode contains a single, 16-bit word integer, which corresponds exactly to a TurboIMAGE DBDELETE *mode* value of 1. Mode 1 deletes the current entry from a manual master or detail data set. Then, it automatically updates the OMNIDEX indexes to remove the sorted key values and OMNIDEX keywords associated with that record.

Mode options

Mode options are integer values that are added to the mode value (1) to elicit the following:

100 **IMAGE-only mode.** The current entry is deleted. OMNIDEX indexes are not changed.

200 **Index-only mode.** Rereads the current entry and removes the key values and keyword occurrences associated with that record without deleting the entry itself.

status is the name of an array of 21 16-bit words used to return information about the success of a call. See Table 2-1, on page 2-7, for a list of error condition codes.

Discussion

In normal mode, DBIDELETE calls DBDELETE to delete the desired entry, then updates any associated indexes as necessary. When a database has been enabled for Third Party Indexing, DBDELETE calls DBIDELETE to update OMNIDEX indexes after a TurboIMAGE record is deleted.

If an error occurs during an index update, a nonzero number is returned to *status* word 1, as described previously in "OMNIDEX status array values", on page 2-6. An abnormal termination of a calling application during a delete could cause an indexing error.

Indexing errors are easy to correct by performing an OmniUtil "Reindex Specific Tables" operation on the affected OMNIDEX data set or domain.

OMNIDEX condition word values

Exceptional conditions

496	ILCB damaged or OLCB damaged
497	Bad base ID, or database not opened using DBIOPEN
4*nn*	Any other 400 level value: IMSAM internal error

Calling errors

-400	Illegal mode specified

DBIERROR

DBIERROR (*status, buffer, length*)

DBIERROR interprets the TurboIMAGE and OMNIDEX condition words and places an error message in the *buffer* parameter.

Parameters

status is the name of an array of 21 16-bit words used to return information about the success of a call. See Table 2-1 on, page 2-7, for a list of error condition codes.

buffer is a 36 16-bit word (72 byte) array where the TurboIMAGE or OMNIDEX error message is returned.

length is a 16-bit word integer containing the byte length of the message, up to 72 characters.

Discussion

DBIERROR interprets TurboIMAGE errors by calling the TurboIMAGE DBERROR intrinsic internally. OMNIDEX errors are interpreted directly by DBIERROR.

TurboIMAGE errors comprise calling errors and exceptional conditions, which can result from a call to TurboIMAGE by an OMNIDEX intrinsic. For example, using DBIPUT to add an entry to a data set that is inaccessible to your user class would result in a TurboIMAGE error. The failing OMNIDEX intrinsic returns nonzero values to the TurboIMAGE condition word and the OMNIDEX condition word of *status*. If the application calls DBIERROR, it in turn calls DBERROR to interpret TurboIMAGE errors.

Sorted access and general intrinsic calling errors and exceptional conditions cause the TurboIMAGE condition word (word 1) to be set to +999. Keyword access calling errors and exceptional conditions cause the TurboIMAGE condition word to be set to +888. In either case, *status* word 11 is set to the value (code) that identifies the OMNIDEX error.

For example, a beginning of file error for a descending sequential read would result in an OMNIDEX exceptional condition. Attempting a partial-key retrieval on an item that isn't a sorted key would result in an OMNIDEX calling error.

A list of OMNIDEX error codes is included with each intrinsic's description. Consult the *TurboIMAGE/XL Database Management System Reference Manual* for TurboIMAGE error codes.

DBIEXPLAIN

DBIEXPLAIN (*status, parm*)

DBIEXPLAIN interprets the TurboIMAGE and OMNIDEX error codes and prints an error message to $STDLIST. At the end of the message, the value of *parm* is displayed after the words, `Program Error`. Note that *parm* is an additional parameter not used by TurboIMAGE's DBEXPLAIN.

Parameters

status is the name of an array of 21 16-bit words used to return information about the success of a call. See Table 2-1, on page 2-7, for a list of error condition codes.

parm is a 16-bit word integer. The value of *parm* determines what happens when there is an error, as listed below.

- =0 suppresses the `Program Error` <*parm*> message. Only the TurboIMAGE or OMNIDEX error message is displayed.
- <0 aborts the user process immediately after displaying the `Program Error` <*parm*> and error messages.
- >0 displays the `Program Error` <*parm*> and error messages, but does not abort the program.

Discussion

Note that you can use different values for *parm* on different intrinsic calls when debugging a program. This helps to identify which call is causing problems.

OMNIDEX calling errors and exceptional conditions are explained by a one or two line message. For errors that occur when an OMNIDEX intrinsic calls a TurboIMAGE intrinsic, DBIEXPLAIN calls DBEXPLAIN to interpret the error.

Errors that occur when an OMNIDEX intrinsic calls a TurboIMAGE intrinsic are first interpreted using DBEXPLAIN. Then the OMNIDEX internal error code is displayed.

DBIEXPLAIN calls DBEXPLAIN to interpret errors whenever necessary. Therefore, it is best to call DBIEXPLAIN immediately after returning from the failing intrinsic to ensure the display of a valid error message.

DBIFIND

DBIFIND (*base, dset, mode, status, item, argument*)

DBIFIND locates a detail chain head using a sorted key installed on the TurboIMAGE search item (SI) of a master set. The SI is specified by the *item* parameter. The detail set is specified by the *dset* parameter.

DBIFIND sets up pointers in preparation for DBIGET chained access. A chain consists of detail entries that share the same value for the SI specified in *item*. The first detail record whose SI values matches the *argument* value is the first chain head. You can locate the chain head by full key value in mode 1, or by partial key value in sorted-key sequence, using modes 100-500. The *argument* parameter contains the full or partial-key value that is used to locate the chain head.

Index-only mode access can be used for all relational and sequential retrieval modes by adding 1000 to the mode value. For more information about index-only mode, see "Index-only mode", on page 2-26.

Parameters

base is the name of the integer array used as the *base* parameter when opening the database. The first element of the array must contain the base ID returned by DBIOPEN. (See page 2-44 for more information about the base ID.)

dset is the name of an array that contains the left-justified name, or the 16-bit set number, of the data set being accessed. The data set name may be up to 16 characters long or, if shorter, terminated by a semicolon (;) or a space (for example, CUSTOMERS; or ORDER-LINES).

mode is a 16-bit word integer, used to pass any of the following values:

TurboIMAGE modes

1 * same as TurboIMAGE DBFIND mode 1 with TPI disabled. Locates the chain head that matches the full key value in *argument*.

Relational modes

To perform a relational operation, add one of the mode values on the next page to the key length of the value passed in the *argument* parameter.

100 **= operation**. Locates the first chain head in ascending key order whose key value equals the value in *argument* to the precision specified. For index-only mode, use 1100.

200 **> operation**. Locates the first chain head in ascending key order whose key value is greater than the specified argument. For index-only mode, use 1200.

300 **>= operation**. Locates the first chain head in ascending key order whose key value is greater than or equal to the specified argument. For index-only mode, use 1300.

400 **< operation**. Locates the first chain head in descending key order whose key value is less than the specified argument. For index-only mode, use 1400.

500 **<= operation**. Locates the first chain head in descending key order whose key value is less than or equal to the specified argument. For index-only mode, use 1500.

Sequential modes

After a call to DBIFIND using a relational mode, these modes let you read through key values. The *argument* parameter is ignored for these modes.

90 locates the next higher SI value.

91 locates the next lower SI value.

92 locates the current chain head. Resets the pointer to the beginning of the chain.

Mode options

1000 add to the mode value to specify index-only mode

status is the name of an array of twenty-one 16-bit words used to return information about the success of a call. See Table 2-1, on page 2-7, for a list of error condition codes.

item is an array of eight 16-bit words, which contain the left-justified name of a TurboIMAGE SI. For all *mode* values except 1, the referenced SI must be a sorted key.

argument passes the full, or partial, key value used to locate the chain head for relational retrieval modes. Returns the retrieved key value for index-only mode.

Discussion

Relational mode values and *argument* length

For DBIFIND mode 1, the *argument* parameter must contain a full key value as a search argument.

For the relational modes (100-500), the *argument* parameter can contain a partial key value. The length of *argument* is expressed in *mode*, as discussed below, along with the relational condition. The relational condition specified in *mode* is limited to the partial key argument. For example, if you supply a partial key argument like 2392A, and reflect its length through *mode*, DBIFIND compares the first five characters of the indexed key values against that partial key argument. The chain head found is the first key value that satisfies the relation to the *argument* parameter, expressed by the relational mode value.

When using a full key argument value (one equal to the defined length of the sorted key), add nothing to the relational mode value. For example, to specify a "greater than" relational operation using a full key argument, the *mode* would be 200. You may need to pad the argument value with trailing blanks to make it a full key value. For example, when searching an X14 PRODUCT-NO sorted key for detail records that contain 2392A, you would pad that value with nine trailing blanks (2392A), to make it 14 characters long.

When using a partial key value in *argument*, add its length to the relational mode value. *argument*'s length can be passed in either words or bytes.

- To express the length in words, add the number of words to the relational mode value and use a positive *mode* value.
- To express the length in bytes, add the number of bytes to the relational mode value and use a negative *mode* value.

For example, to specify a two word *argument* value (like 2392) in a greater than or equal to operation (mode 300), use 302 for the *mode*. To specify a five-byte argument (like 2392A), use -305 for the *mode*.

When using partial key lengths of 100 bytes or longer, specify the partial key length in words. If you try to add a partial-key length greater than 100 bytes to a mode value, it results in the next higher mode. For example, a relational mode of 100 (equal to) with a partial-key length of 102 bytes results in a mode of -202, which is a greater than operation with a byte length of "2" instead of an equal to operation with a byte length of "102".

Therefore, if you have sorted keys longer than 100 bytes, use partial-key values on word boundaries. Using the example above, a mode of 100 plus a partial-key value of 51 words results in a mode of 151, which yields the desired retrieval.

More examples of mode values for partially specified arguments are included with the DBIGET command description.

Index-only mode

An index-only mode DBIFIND differs from a normal mode DBIFIND in two respects:

- An index-only mode DBIFIND retrieves the first key value that qualifies, and returns that key value to the calling program via the *argument* parameter, overwriting the value that was passed in the call. A normal mode DBIFIND uses the first key value that qualifies in a call to DBFIND, and does not alter the *argument* parameter.
- An index-only mode DBIFIND does not set up pointers for chained access (DBIGET mode 5 or 6) to the detail set, but a normal mode DBIFIND does.

To specify index-only mode, add 1000 to the relational or sequential mode value. Some examples are: 1090, 1091, 1100, 1302 and -1505.

Index-only mode calls to DBIFIND can be used to retrieve all key values that match a partial value entered by a user in an application program. For example, the first DBIFIND does a partial-key retrieval, usually in an equals operation, to retrieve the first key that qualifies. Then index-only sequential mode (1090) calls to DBIFIND are performed to retrieve subsequent SI values in sorted order.

OMNIDEX condition word values

Exceptional conditions

201	Operation stopped by user
210	Beginning of file
211	End of file
213	IMSAM tree empty
217	Key not found
296	ILCB damaged
297	Bad base ID, or database not opened using DBIOPEN
2*nn*	Any other 200 level value: IMSAM or OMNIDEX internal error or TurboIMAGE error

Calling errors

-200	Illegal mode specified
-201	Data set not an IMSAM detail
-202	Key value exceeds defined key length
-204	Item not an IMSAM key
-212	No current key

DBIGET

DBIGET (*base, dset, mode, status, list, buffer, argument*)

DBIGET provides several different methods for retrieving all or part of a record via a sorted key. You can also call DBIGET to perform standard TurboIMAGE retrievals.

Use DBIFIND to find detail chains via the SI of a master set.

Parameters

base is the name of the integer array used as the *base* parameter when opening the database. The first element of the array must contain the base ID returned by DBIOPEN. (See page 2-44 for more information about the base ID.)

dset is the name of an array of 16 16-bit words (32-bytes) that contains the left-justified name or number of the data set to be accessed and the name or number of the sorted key. Note that this differs from the standard TurboIMAGE *dset* parameter.

The data set name must be in the first 8 words (16 bytes), and the sorted key name must begin in word 9 (byte 17). If the data set name is shorter than 8 words, it must be terminated by a semicolon (;) or a space.

If a master set's search item is the only sorted key defined in the set, the last 8 words (16 bytes) are ignored.

If a TurboIMAGE mode is used, the last 8 words (16 bytes) are ignored.

mode is a 16-bit word integer, that contains any of the following values:

TurboIMAGE modes

1 * reread. Standard DBGET mode 1.

2 * serial read. Standard DBGET mode 2.

3 * backward serial read. Standard DBGET mode 3.

4 * directed read. Standard DBGET mode 4.

5 * chained read. Standard DBGET mode 5.

6 * backward chained read. Standard DBGET mode 6.

7 * calculated read. Standard DBGET mode 7.

8 * primary calculated read. Standard DBGET mode 8.

Relational modes

Base values to which the number of bytes or words are added.

100 **= operation**. Retrieves the first entry in ascending key order whose key equals the specified key value. For index-only mode, use 1100.

200 **>operation**. Retrieves the first entry in ascending key order whose key is greater than the specified key value. For index-only mode, use 1200.

300 **>= operation**. Retrieves the first entry in ascending key order whose key is greater than or equal to the specified key value. For index-only mode, use 1300.

400 **< operation**. Retrieves the first entry in descending key order whose key is less than the specified key value. For index-only mode, use 1400.

500 **<= operation**. Retrieves the first entry in descending key order whose key is less than or equal to the specified key value. For index-only mode, use 1500.

Sequential modes

90 retrieves the next entry in ascending key sequence. The *argument* parameter is ignored. For index-only mode, use 1090.

91 retrieves the previous entry in ascending key sequence (the first entry in descending order). The *argument* parameter is ignored. For index-only mode, use 1091.

92 reread the current entry. The *argument* parameter is ignored. For index-only mode, use 1092.

Mode options

1000 specifies index-only mode when added to a mode value.

status is the name of an array of 21 16-bit words used to return information about the success of a call. See Table 2-1, on page 2-7, for a list of error condition codes.

list is the name of an array that contains either an ordered set of data item names or numbers, an at sign (@) to specify all items in the database, or an asterisk (*) to re-specify items passed in the last *list*.

See the *TurboIMAGE/XL Database Management System Reference Manual* discussion of DBGET for more information about the *list* parameter.

buffer is the name of the array to which the values of data items specified in the *list* array are moved. The defined array must be large enough to hold the values from the fields that correspond to the items in *list*. If the *buffer* array is to small, a memory error will result.

argument contains the full or partial key value used to locate the master or detail record for all relational retrieval modes.

Discussion

DBIGET is used for retrieval on all sorted keys, except when locating chain heads via sorted keys installed on TurboIMAGE search items. To locate chain heads and perform chained reads, use DBIFIND instead.

Relational mode values and *argument* length

For all relational retrieval modes, add the length of any partial key value supplied for the *argument* parameter to the relational mode value.

When using a full key *argument* value (one equal to the defined length of the sorted key) with a relational mode value, add nothing to the *mode* value. For example, to specify a "greater than" relational operation using a full key *argument* value, the *mode* value would be 200. You may need to pad an argument with trailing blanks to make it a full key value. For example, when searching an X14 PRODUCT-NO sorted key for records that contain 2392A, you would pad the argument with nine trailing blanks (2392A).

The length of a partial key value contained in the *argument* parameter can be passed in either words or bytes.

- To express the length in words, add the number of words to the relational mode value and use a positive *mode* value.
- To express the length in bytes, add the number of bytes to the relational mode value and use a negative *mode* value.

For example, to specify a two-byte (one word) key value, like HE, in a greater than or equal to operation (mode 300), you would use 301 for the length in words or -302 for the length in bytes.

The scope of the relational search is limited to the key length specified in *mode*. If you are searching an X14 key, and the *mode* value is 102, DBIGET retrieves the record of the first key value whose first (two words) four bytes match the argument.

Some examples of this follow.

- Get the first entry whose key value starts with "P". *Argument* length is 1 byte and the relational mode is 100, therefore: *mode* = -101.
- Get the first entry whose key value is greater than "MA" in the first two bytes. *Argument* length is 2 bytes or 1 word and the relational mode is 200, therefore: *mode* = -202 or 201.
- Get the highest key value that is less than or equal to "R5A". *Argument* length is 3 bytes and the relational mode is 500, add 1000 for index-only mode (see below), therefore: *mode* = -1503.
- Get the first entry whose key value is greater than "Smith, John Q.", assuming the key's length is 14 bytes. *Argument* length is 14 bytes or 7 words, or the full defined length, and the relational mode is 200, therefore *mode* = -214 or 207 or 200.

For sorted keys whose total length is equal to or greater than 100 bytes (including the search item or relative record number for index-only mode), specify partial key lengths on word boundaries, as discussed on page 2-25.

Normal mode and index-only mode

In normal mode, a record is retrieved and the field values specified in the *list* parameter are returned to the calling program.

In index-only mode, DBIGET retrieves only the key values from an index. No record is accessed. Index-only mode is available for all relational and sequential retrieval modes by adding 1000 to the mode value. Use an index-only relational mode to retrieve the first key value, and index-only sequential modes to retrieve subsequent key values.

In index-only mode, the *list* parameter is ignored and the retrieved key value is returned to the calling program via the *buffer* parameter. It has the following format:

Word	Contents of *buffer* parameter
1 — *n*	key value, where *n* = key length in words
n — *n*+*ID*	ID value, where *ID* represents an SI value (for masters) or a 4-byte relative record number (for details).

If the key is a sorted-only key (not also a TurboIMAGE search item), the full key value is returned. The full key is the sorted key value plus the record's search item (for a master data set) or the relative record number (for a detail data set). See page 2-26 for some applications of index-only mode.

Programming tips

For TurboIMAGE search items that are also sorted keys, if the full key value is known and a relational equals operation is desired, use TurboIMAGE mode 7 instead of mode 100. Mode 7 (TurboIMAGE calculated access) eliminates a needless index search to locate the key value.

To find the highest value for a key, use DBIGET mode 500 (less than or equal to) with a high *argument* value.

When using partial key values or ranges of key values, you can perform mode 90 or 91 calls to DBIGET to retrieve records (or key values) in sequence. However, you must provide a programmatic check to determine when the records retrieved no longer match the partial key value, or are out of range.

OMNIDEX condition word values

Exceptional conditions

301	Critical flag set
310	Beginning of file
311	End of file
313	Index file empty
314	No keys in the specified range
317	Key not found
347	Index file has wrong data format for this computer system
396	ILCB damaged
397	Bad base ID, or database not opened using DBIOPEN
3*nn*	Any other 300 level value: IMSAM or OMNIDEX internal error or TurboIMAGE error

Calling errors

-300 Illegal mode specified

-301 Data set not an IMSAM data set

-302 Key value exceeds defined key length

-304 Item not an IMSAM key

-305 DBIFIND followed by chained DBIGETs required for TurboIMAGE search items

-312 No current value for this key

-315 Not an IMSAM database

DBIINFO

DBIINFO (*base, qualifier, mode, status, buffer*)

DBIINFO provides information about a database. This includes:

- keys installed on a data set
- data type of a particular item
- type of access permitted for each item
- information about Intrinsic Level Recovery

Parameters

base is the name of the integer array used as the *base* parameter when opening the database. The first element of the array must contain the base ID returned by DBIOPEN. (See page 2-44 for more information about the base ID.)

qualifier is the name of an array of 8 16-bit words (16 bytes), which contains a data set name or number or a sorted key's item number.

In mode 312, *qualifier* is an array of 16 16-bit words (32 bytes), which contains a data set name or number followed by a sorted key name or number starting at word 9 (byte 17). See the tables which follow.

mode is a 16-bit word integer that specifies the type of information desired.

DBIINFO can be used with the TurboIMAGE modes and functions almost identically to the corresponding TurboIMAGE DBINFO call. There are four (4) additional modes that are used by OMNIDEX only.

status is the name of an array of 21 16-bit words used to return information about the success of a call. See Table 2-1, on page 2-7, for a list of error condition codes.

Word 2 contains the number of words of information returned to the *buffer* parameter.

buffer is the name of the array that contains the returned information.

The following pages list each mode, its purpose, and the contents of the *qualifier* and *buffer* parameters.

mode value	Purpose	*qualifier* contents	*buffer* element	array contents	Comments
101 *	Returns the type of access available for an item	data item name or number	1	±data item number	If negative, write access is available to the item in at least one set. Item numbers > 2000 indicate a composite sorted key. Item numbers > 1024 indicate a composite keyword key.
102 *	Returns description of data item	data item name or number	1 . . 8 9 10 11 12 13	data item name data type sub-item length sub-item count zero zero	Left-justified and padded with blanks if required I, J, K, R, U, X, Z, P. CO for composite keyword key. CI for composite sorted key
103 *	Identifies all items available in database and type of access supported	(ignored)	1 2 . . *n*+1	number (*n*) of data items available ±data item number for each item in the database	Size of array depends on number of items. Items are arranged in item definition order. Negative item number indicates that the item is write accessible in at least one set. Item numbers > 2000 indicate a composite sorted key. Item numbers > 1024 indicate a composite keyword key.
104 *	Identifies all items available in data set referenced in *buffer*, and type of access supported	data set name or number	(same as mode 103)		Item numbers are listed in the order that they occur in the data set. Information returned by item numbers is the same as for mode 103.

Table 2-2: DBIINFO mode values

mode value	Purpose	*qualifier* contents	*buffer* element	array contents	Comments
201 *	Defines type of access available for the referenced data set	data set name or number	1	±data set number	If negative, write access is permitted.
202 *	Describes the referenced data set	data set name or number	1 . . 8	data set name	left-justified and padded with blanks if required.
			9	set type abbreviation	**M**anual master; **D**etail; **A**uto master.
			10	entry-word length	Words 10 and 11 are integer values.
			11	blocking factor	
			12	sorted key information	**I** means sorted keys on data items. **M** means master with sorted key on search item. **A** means auto master with sorted key on search item.
			13	zero	
			14	number of entries in set	Words 14 through 17 are integer values.
			15		
			16	capacity of set	
			17		
203 *	Identifies all data sets available in the database and the type of access supported for each	(ignored)	1	number of sets available (*n*).	
			2 . . . *n* +1	±data set number . . . ±data set number	Data set numbers are returned in order. Negative number means write access. Positive means read and possibly update access.

Table 2-2: DBIINFO mode values

mode value	Purpose	*qualifier* contents	*buffer* array contents element		Comments
204 *	Identifies all data sets that contain the referenced data item	data item name or number	(same as mode 203)		(same as mode 203)
301 *	Identifies the paths defined for the referenced data set	data set name or number	1 2 3 4 . . . . 3*n*-1 3*n* 3*n*+1	number of paths (*n*) ±data set number SI number sort item number . . . . ±data set number SI number sort item number	If set number is negative, write access is permitted for that set. If set referenced in *qualifier* is a master, the set in element 2 is a detail. If referenced set is a detail, the set in element 2 is a master. Item numbers in elements 3 and 4 refer to items in detail. Words 2-4 repeat for each path. Path designators are listed in order of their appearance in the schema.
302 *	Describes referenced data set	master data set name or number OR detail data set name or number	1 2 OR 1 2	SI number zero OR primary path item number data set number of related master set	This is zero if the SI is inaccessible Words 1 and 2 are zero if the SI is inaccessible
310	Identifies all IMSAM data sets	(ignored)	1 2 . . . *n*+1	number of IMSAM sets (*n*) set number set number	(same as mode 203)

Table 2-2: DBIINFO mode values

mode value	Purpose	*qualifier* contents	*buffer* element	array contents	Comments
311	Identifies sorted keys in referenced data set	data set name (UPPER CASE) or number (16 bit integer)	1 2 . . . *n*+1	number of sorted keys (*n*) item number of key item number of key	 An item number greater than 2000 indicates a composite sorted key. This element repeats *n* times, once for each key.
312	Returns information about referenced sorted key	32 byte array. First 16 bytes contain data set name (UPPER CASE) or number (16 bit integer). Bytes 17-32 contain sorted key name or item number.	1 2 3 4 5 6 . . *n* +4	item number of key byte 1: installation type byte 2: bit map of key options key length (in bytes) number of components (*n*) byte offset length of each component	 Byte 1: **I** means sorted key on data items. **M** means sorted key on master's search item. **A** means sorted key on auto master's search item. Byte 2: 8:1 Batch indexing 9:1 No Translate 10:1 No Exclude For sorted keys on data items in a master, contains byte offset and length of SI. For sorted keys on data items in a detail contains two zeros. Elements 5 and 6 are repeated for as many components as are counted in element 4.
313	Identifies sorted keys	item number	1 . . . 8 9	sorted key's name key type	 CI means composite sorted CO means composite keyword
320	Returns number of IMSAM keys in Base	none	1	number of indexes	

Table 2-2: DBIINFO mode values

mode value	Purpose	*qualifier* contents	*buffer* element	array contents	Comments
321	Identifies version of intrinsics	name (UPPER CASE) or number (16-bit integer) of data set that contains a sorted key	1 2	version number of the intrinsics	for example 30408 means version 3 release 04 fix level 08
401 *	Returns logging information	(ignored)	1 . . 4	log identifier name	left-justified and padded with blanks if required
			5	database log flag	1 if logging is enabled 0 if logging is disabled.
			6	user log flag	1 means user is logging.
			7	transaction log flag	1 means user has a transaction in progress.
			8 9	user transaction number	
402 *	Returns ILR information	(ignored)	1	ILR log flag	1 if ILR enabled for database.
			2	calendar date	Date ILR enabled (*mmddyy*)
			3	clock time	Time ILR enabled (*hhmmsstt*)
			4 5 6 . . . 14	zero blank	
			15 16	reserved	

Table 2-2: DBIINFO mode values

DBILOCK

DBILOCK (*base, qualifier, mode, status*)

DBILOCK calls DBLOCK to lock entries, data sets or the database, depending upon the *qualifier* parameter. DBILOCK is provided only for backward compatibility with programs written for earlier versions of OMNIDEX.

Parameters

base is the name of the integer array used as the *base* parameter when opening the database. The first element of the array must contain the base ID returned by DBIOPEN. (See page 2-44 for more information about the base ID.)

qualifier is the name of an array that contains a data set number, a data set name or a set of lock descriptors, depending on the mode value, as discussed below:

Modes 1 and 2	ignored
Modes 3 and 4	a 16-bit integer variable referencing the data set number, or the name of an eight 16-bit word (16 byte) array containing a data set name. Could also be @ to apply a base level lock.
Modes 5 and 6	the name of the array that contains the lock descriptors. The format for lock descriptors is discussed in the *TurboIMAGE/XL Database Management System Reference Manual*.

Be careful when changing modes; the *qualifier* parameter may also change.

mode is a 16-bit word integer that specifies the type of locking through one of the following values:

1 *	base level, unconditional locking
2 *	base level, conditional locking
3 *	set level, unconditional locking
4 *	set level, conditional locking
5 *	item (entry) level, unconditional locking
6 *	item (entry) level, conditional locking

status is the name of an array of twenty-one 16-bit words used to return information about the success of a call. See Table 2-1, on page 2-7, for a list of error condition codes.

Discussion

DBILOCK is provided so that OMNIDEX version 3 is backwardly compatible with programs written under previous versions of OMNIDEX. OMNIDEX indexes are locked automatically during puts, deletes, and updates. This is similar to Optimized Locking under versions 2.05 through 2.10.

Because OMNIDEX index files must be locked during updates, a program must have Multi-RIN (MR) capability if it calls DBIPUT, DBIUPDATE, or DBIDELETE after a call to DBLOCK or DBILOCK.

DBIOPEN

DBIOPEN (*base, password, mode, status*)

DBIOPEN initiates access to a database and establishes the user class number and access mode for all subsequent database access.

Parameters

base is the name of an integer array that contains a string of ASCII characters. The string must contain two leading spaces, followed by a left-justified database name, and terminated by a semicolon (;) or space, for example, SALES; or SALES . The database can be qualified to the group and account level, for example, SALES.DEMODB.DISC;.

If the database is successfully opened, TurboIMAGE replaces the two leading spaces with a value called the *base ID*. The base ID uniquely identifies the access path between the database and the process calling DBIOPEN. In all subsequent accesses to the database the first word of ***base*** must be this base ID; therefore, the array should not be modified for subsequent calls to intrinsics that use a ***base*** parameter.

password is the name of a 16-bit word array that contains a left-justified string of ASCII characters of an optional password followed by an optional user identifier.

mode is a 16-bit word integer from 1 to 8, identical to TurboIMAGE's DBOPEN modes.

1 *	MODIFY with enforced locking. Allow concurrent modify
2 *	UPDATE, allow concurrent update
3 *	MODIFY exclusive
4 *	MODIFY, allow concurrent read
5 *	READ, allow concurrent access in modes 1 or 5
6 *	READ, allow concurrent access in modes 2, 4, 6, or 8
7 *	READ, exclusive
8 *	READ, allow concurrent read

status is the name of an array of twenty-one 16-bit words used to return information about the success of a call. See Table 2-1, on page 2-7, for a list of error condition codes.

OMNIDEX condition word values

Exceptional Conditions

901	Rootfile corrupt
904	OMNIDEX expired
905	IMSAM expired (IMSAM error message) or OMNIDEX expired (OMNIDEX error message)
912	ILCB stack overflow - too many databases open
913	Index inaccessible
941	Memory allocation failure - cannot create control blocks
949	Not a current version 3 rootfile. Reinstall IMSAM/OMNIDEX
9*nn*	IMSAM or OMNIDEX internal error or TurboIMAGE error

DBIPUT

DBIPUT (*base, dset, mode, status, list, buffer*)

DBIPUT adds entries to the data set specified in *dset*, placing the data that is in the *buffer* parameter into the fields specified in *list*.

Parameters

base is the name of the integer array used as the *base* parameter when opening the database. The first element of the array must contain the base ID returned by DBIOPEN. (See page 2-44 for more information about the base ID.)

dset is the name of an array that contains the left-justified name, or the 16-bit number, of the data set being accessed. The data set name may be up to 16 characters long or, if shorter, terminated by a semicolon (;) or a space (for example, CUSTOMERS; or ORDER-LINES).

mode is a 16-bit word integer value of 1. It adds an entry to a manual master or detail data set. Any IMSAM or OMNIDEX indexes associated with the entry are updated automatically.

Mode options

An integer value which when added to the mode elicits the following:

100 **IMAGE-only mode**. OMNIDEX indexes are not changed.

200 **Index-only mode**. Indexes the OMNIDEX keys for an entry without adding the entry itself. An @ item list must be used or all fields must be specified in set definition order.

status is the name of an array of twenty-one 16-bit words used to return information about the success of a call. See Table 2-1, on page 2-7, for a list of error condition codes.

list is the name of an array, also called the *item list*, that contains an ordered set of data item identifiers, which can be names, numbers, or an at sign (@).

For most data sets that contain OMNIDEX keys, an @ list or the equivalent (all item numbers or names in set definition order) must be used. The exceptions are sets in Transparent domains, or unlinked (Detail Record indexed) detail sets, which need not use an @ item list. Use ODXINFO modes 7 and 8 to determine the presence of a Transparent ID or Detail Record indexing for a set.

buffer is the name of an array that contains the data to be added as an entry.

Discussion

OMNIDEX ID assignment

When using DBIPUT mode 1 to add a record to a master data set (table) that uses an I2, J2, or K2 search item, you can let it assign the integer SI value for you. To assign the *next free ID*, when adding the record, supply a value of -1 in the area of *buffer* that corresponds to the SI/ID field. The next available ID is the lowest ID value that once belonged to a deleted master record. Note that if no such IDs are available, DBIPUT assigns the next unused ID, as discussed below.

To assign the *next unused ID*, supply a value of 0 (zero) in the area of *buffer* that corresponds to the SI/ID field. The next unused ID is the ID value that is one higher than the highest used value. For example, if the highest (integer) ID value for the customers set is 5878972, the highest unused value is 5878973.

Abnormal termination

If a system failure or program abort occurs while DBIPUT is adding a key value to an index, the index may not be updated. It is structurally intact, but it is missing a key value.

In normal mode, DBIPUT first adds the desired entry, then updates the associated indexes. Thus, if the abnormal termination occurred after the entry was added but before the index was modified, the index would be corrupted. This situation occurs very rarely and is easy to correct by reindexing the affected table or domain using OmniUtil.

OMNIDEX condition word values

Exceptional conditions

105 FLOCK failure; program may lack MR capability

141 IMSAM tree full

196 ILCB damaged or OLCB damaged

197 Bad base ID, or database not opened using DBIOPEN

Calling errors

-100 Illegal mode specified

-101 An @ item list required in index-only mode

-114 OMNIDEX ID must be greater than 0

-119 Deferred updates restricted to one domain

DBIUNLOCK

DBIUNLOCK (*base, dset, mode, status*)

DBIUNLOCK releases all locks set by previous calls to DBLOCK or DBILOCK against the database identified in *base*.

Parameters

base is the name of the integer array used as the *base* parameter when opening the database. The first element of the array must contain the base ID returned by DBIOPEN. (See page 2-44 for more information about the base ID.)

dset is ignored.

mode must be a 16-bit word integer value of 1.

status is the name of an array of twenty-one 16-bit words used to return information about the success of a call. See Table 2-1, on page 2-7, for a list of error condition codes.

DBIUPDATE

DBIUPDATE (*base, dset, mode, status, list, buffer*)

DBIUPDATE updates the current entry in the specified data set, placing the data that is contained in the *buffer* array into the fields specified by the *list* parameter. It also updates any indexes to reflect the updates to TurboIMAGE fields.

Parameters

base is the name of the integer array used as the *base* parameter when opening the database. The first element of the array must contain the base ID returned by DBIOPEN. (See page 2-44 for more information about the base ID.)

dset is the name of an array that contains the left-justified name, or the 16-bit number, of the data set being accessed. The data set name may be up to 16 characters long or, if shorter, terminated by a semicolon (;) or a space (for example, CUSTOMERS; or ORDER-LINES).

mode is a 16-bit word integer value of 1. It updates the current entry, then updates the OMNIDEX indexes as necessary.

Mode options

An integer value that you can add to the mode to elicit the following:

100 **IMAGE-only mode**. OMNIDEX indexes are not changed.

status is the name of an array of twenty-one 16-bit words used to return information about the success of a call. See Table 2-1, on page 2-7, for a list of error condition codes.

list is the name of an array that contains an ordered list of data item identifiers. These can be either names or numbers, an asterisk (*), or an at sign (@).

If the structure of the database you are updating was not altered before installing OMNIDEX, you may need to change programs that open a database in mode 3 to use an "@" item list, or the equivalent, in calls to DBIUPDATE. This is necessary for most sets that have OMNIDEX keys. Master sets (tables) with a character search item, and unlinked detail sets are the exceptions.

buffer is the name of an array that contains the data item values to be added.

Discussion

Use DBIUPDATE to update records in data sets that contain OMNIDEX keys. DBIUPDATE automatically updates any indexes affected when data is changed in an indexed field (a keyed field, or component in a composite key). It is recommended for updates on all data sets. Note that DBIUPDATE does not support critical item updates. That is, if you change TurboIMAGE search item values, OMNIDEX keys will not be indexed correctly.

OMNIDEX condition word values

Calling errors

-612 OMNIDEX SI cannot be modified

Keyword Access Intrinsics

The Keyword access intrinsics are:

- ODXFIND
- ODXGET
- ODXGETWORD
- ODXINFO
- ODXPRINT
- ODXTRANSFER
- ODXVIEW

These intrinsics are similar to TurboIMAGE intrinsics in terms of the parameters they use. ODXFIND qualifies records based on any combination of keywords (words and values) for a given keyword key or group of keys. ODXGET retrieves the qualified records' TurboIMAGE search values item or relative record numbers. Then, DBFIND and/or DBGET are used to retrieve the qualifying records.

Keyword retrieval

The ODXFIND and ODXGET intrinsics provide keyword searches on keyword keys. When you install a keyword key on a field, an index is created for that key (and other keys in the same *OMNIDEX domain*). When the index is loaded, data values are parsed into keywords, based on the spaces (or non-alphanumeric characters) that separate them.

Use ODXFIND to search the index for records that contain keyword arguments. Then, use ODXGET to retrieve the search item values for those records. Finally, use DBGET in mode 7 or mode 4, or DBFIND followed by DBGET mode 5 to retrieve the actual records.

The Keyword access intrinsics support a variety of search operations using either fully specified, or partial (generic) keyword arguments. Among the operations supported are:

- Boolean (AND, OR, and NOT) operations
- relational (<, <=, =, >=, >) operations
- range (*startvalue:stopvalue*) operations

For more information about keyword searches, see ODXFIND, discussed next, or "OMNIDEX keyword retrieval," on page 3-13 of the "Programming" chapter.

ODXFIND

ODXFIND (*base, dset, mode, status, field, keywords*)

ODXFIND searches the indexes for keywords and the records that contain them. It returns the SIs, relative record numbers, or just keywords, depending on the type of domain being searched, and the *mode* value used. It stores these values in memory pending a call to ODXGET, or ODXGETWORD.

Parameters

base is the name of the integer array used as the *base* parameter when opening the database. The first element of the array must contain the base ID returned by DBIOPEN. (See page 2-44 for more information about the base ID.)

dset is the name of an array that contains the left-justified name, or the 16-bit number, of the data set being accessed. The data set name may be up to 16 characters long or, if shorter, terminated by a semicolon (;) or a space (for example, CUSTOMERS; or ORDER-LINES).

mode is a single, 16-bit word integer. Modes 1, 2, 3, 5, and 30 are used before an ODXGET; modes 10 and 11 are used before an ODXGETWORD intrinsic call.

- 1 returns the number of qualifying OMNIDEX ID values based on the arguments and operations passed in the *keywords* parameter.
- 2 same as mode 1.
- 3 enables the parsing of literal and parenthetical operators in the *keywords* parameter. When mode 3 is specified, the keyword list must be terminated by a semicolon (;). See "Modes 3 and 5 enhanced argument parsing", on page 2-58.
- 5 like mode 3, but precedence of operations matches that of the TPI Interface. See "Modes 3 and 5 enhanced argument parsing", on page 2-58.
- 10 returns the number of qualifying keywords within the single range or generic value specified, and sets a pointer to the first qualifying keyword.

11 sets a pointer to the first qualifying keyword, as in mode 10, but does not return a qualifying count.

30 converts the record IDs of a search on a linked detail set from record-specific (relative record numbers) to record complex (masters' search items). The list of qualifying IDs is compressed, since multiple occurrences of any SI are reduced to a single occurrence, and record number information is discarded.

Mode options

To select mode options, add their values, individually or combined, to the mode value. For example, mode 301 specifies a mode 1 ODXFIND with mode options 100 and 200 enabled.

100 enables **[ctrl]-Y** interrupts during ODXFIND.

200 disables block mode during ODXFIND.

(300) enables **[ctrl]-Y** interrupts, and disables block mode.

400 disables the ID preservation feature on ODXFIND calls that qualify no IDs (see page 2-68).

(500) disables the ID preservation feature and enables **[ctrl]-Y** interrupts.

(600) disables block mode and the back-out feature.

(700) enables **[ctrl]-Y** interrupts, disables block mode and the back-out feature.

Both mode options 100 and 200 (mode option 300), are recommended with V/3000 to permit [ctrl]-Y interrupts.

status is the name of an array of twenty-one 16-bit words used to return information about the success of a call. See Table 2-1 on, page 2-7, for a list of error condition codes. Information is returned to *status* as follows:

Word	Contents of *status* array
1	TurboIMAGE condition word is zero.
2	zero
3-4	in searches on non-Record Complex (record specific) keys in detail sets, contains the number of qualified detail records (represented by their relative record numbers). In searches on keys in master sets, or Record Complex keys in detail sets, contains the number of record complexes (represented by their SIs).
5-6	in most searches, contains the number of qualified masters' search items (a record complex count). In searches on unlinked detail sets, contains the number of qualified detail records (represented by their relative record numbers).
7-8	contains the OMNIDEX ID count for the current search before the IDs are applied to any internal ID list stored in memory
11	OMNIDEX condition word is zero.
12-13	number of qualifying IDs (modes 1, 2, and 30) or number of qualifying keywords (mode 10)
14-15	not used
16-21	contains the first 12 characters of the keyword that caused a calling error (if the call was unsuccessful because of an incorrect word in the *keywords* parameter)

If *status* word 1 is nonzero, word 11 contains the OMNIDEX condition word. DBIERROR or DBIEXPLAIN can be called to interpret the error. See the "OMNIDEX condition word values", on page 2-69, for a list of error numbers and their corresponding error messages.

field is the name of an 8 16-bit word (16 byte) array containing the left-justified name or 16-bit integer item number of a key within *dset*.

All records that contain the keywords specified in the *keywords* parameter for this field qualify. If this field was grouped with others during installation, all the records that contain the specified keywords in this field or any other field in its group qualify.

keywords is the name of an array containing a list of keywords or partial keywords separated by search operators. See page 2-59 for more information about the types of operations supported.

Use a semicolon (;) to terminate the *keywords* list. In mode 1 or 2, you can use a space to terminate the list. Spaces will not terminate the *keywords* list if they are enclosed in quotes. In mode 3 or 5, a space is parsed as an AND operator, so use a semicolon to terminate the *keywords* list.

When ODXFIND is called with mode 3 or 5, the *keywords* list must be terminated by a semicolon (;).

In mode 3 or 5, the *keywords* parameter cannot be longer than 1024 bytes. In mode 1 or 2, the *keywords* parameter cannot be longer than 512 bytes.

Discussion

Modes 1 and 2

Modes 1 and 2 are used in preparation for calls to ODXGET. ODXFIND mode 1 or 2 qualifies record complexes or individual detail records based on a list of keywords, then returns a qualifying count of how many it found. Record complexes are identified by the OMNIDEX search item (SI) of the specified data set. Individual detail records are identified by record number, or by SI value if the Record Complex key option is used. These identifiers are called OMNIDEX IDs.

Modes 3 and 5 enhanced argument parsing

Modes 3 and 5 are used to parse the *keywords* list for advanced retrieval options before a search. Modes 3 and 5 work like mode 1, but enable a user to refine a search through enhanced parsing logic.

ODXFIND modes 3 and 5 support parsing operations that let you:

- use literal operators (like AND, OR, NOT and TO) and spaces (for AND) in a keyword list.
- override the precedence of Boolean operations through parenthetical nesting (like (SOFTWARE AND CONSULT@) OR CONTRACT@).
- search on date fields without having to programmatically translate keyword search values into the correct storage format

Each of these parsing enhancements are described below.

Order of precedence

The only difference between ODXFIND modes 3 and 5 is the order in which they perform operations:

Mode 3	TO, >, <, and so on (range and relational)	OR	NOT (Boolean)	AND
Mode 5	TO, >, <, and so on (range and relational)	NOT	AND (Boolean)	OR

Parenthetical operators

Use parenthetical operators to override the precedence of operations. For example, in mode 3 all OR operations are performed before AND operations. If you wanted to search for all the software consultants and systems engineers, your keyword list would look like this:

```
(SOFTWARE AND CONSULT@) OR (SYSTEM AND ENGINEER@)
```

Without the parentheses, ODXFIND would search for records with the keywords CONSULT@ or SYSTEM, and intersect them with records that contain SOFTWARE and ENGINEER.

Literal operators

ODXFIND modes 3 and 5 parse tokens into equivalent, internal OMNIDEX operators. In the following table, each operation is listed with the equivalent internal operator, and the acceptable literal values. Literal values are not case sensitive.

Operation		Internal OMNIDEX operator	Modes 3 and 5 equivalent tokens (literal operators)
Boolean	and	,	AND or spaces between keywords
Boolean	not	,-	NOT
Boolean	or	+	OR
Range	to	:	TO
Relational	equal to	N/A*	=
Relational	greater than or equal to	N/A	>=
Relational	greater than	N/A	>
Relational	less than or equal to	N/A	<=
Relational	less than	N/A	<

Table 2-3: ODXFIND mode 3 or 5 operations and tokens

* See Table 2-5, on page 2-65, for information about simulating relational retrievals in mode 1 or 2.

ODXFIND mode 1 accepts only the internal operators and does not accept parentheses. ODXFIND modes 3 and 5 accept either the internal operators or their equivalent tokens.

Double quotes

Use double quotes around a keyword value to prevent it from being parsed. This is useful when searching No Parse keys using an argument that contains special characters or spaces (for example, "2392-A") or when an argument could be interpreted as an operator (for example, "OR" in a search on a STATE key).

Date fields

The %DATE function supports the entry and conversion of several different date formats. This feature enables:

- searches and keyword-only searches on proprietary date fields (like ASK, and PowerHouse JDATE and PHDATE date fields)
- consistent entry formats despite inconsistent storage formats
- partial-date arguments (like YYMM, YYYYMM, YY, etc.)

ASKDATE, JDATE, and PHDATE fields should be type cast as type K, if they are not defined as such, to support the %DATE function.

%DATE can be entered as a keyword in a keyword list using the following syntax:

%DATE(*date, entry, storage*)

Each of the parameters are discussed below:

date represents the date argument value. For example, 930509 or 050993.

entry represents the order of the *date* value. The acceptable values for *entry* are: YYMMDD (the default format), YYYYMMDD, YY, YYYY, YYMM, YYYYMM, MMDDYY, MMDDYYYY, MMYY, MMYYYY, DDMMYY and DDMMYYYY.

If no value is entered, the entry format defaults to YYMMDD. This parameter is positional and must be delimited by commas.

storage represents the storage format of the date field on which you are searching. This could be any of the following proprietary formats: PHDATE (PowerHouse internal date format), JDATE (PowerHouse Julian and HP calendar date format) and ASKDATE (ASK Computer Systems internal date format). In addition, the following are supported: YYMMDD, YYYYMMDD, MMDDYY, MMDDYYYY, DDMMYY, DDMMYYYY.

When a partial date *entry* format (i.e., YY, YYYY, YYMM, YYYYMM, MMYY or MMYYYY) is used, the *storage* format must be one of the following: ASKDATE, PHDATE, JDATE, YYMMDD, YYYYMMDD. Also note that partial date entry formats are not supported for range operations.

Conversion of partial dates to any other format besides ASKDATE, PHDATE, JDATE, YYMMDD, YYYYMMDD (like MMDDYY) is not currently supported. To support partial searches on an MMDDYY field, for example, you could create a composite keyword key that reorders the field into YYMMDD format. See the *OMNIDEX ImagePlus SDK Administrator's Guide* for more information.

Below are some examples of the %DATE function's uses.

To convert a YYMMDD date argument into an ASK proprietary date format, you would enter:

```
%DATE(900101,YYMMDD,ASKDATE)
```

You could perform the same search perform without specifying an entry format because it is the default (YYMMDD). Note, however, that double commas are used to pass a null value for the entry format:

```
%DATE(900101,,ASKDATE)
```

To convert a century date entry format into the typical ASCII YYMMDD storage format, you would enter:

```
%DATE(01011990,MMDDYYYY,YYMMDD)
```

As discussed above, the %DATE function also supports partial entry formats. When a partial entry format is used, ODXFIND converts the partial date value into an equivalent range operation. Here, %DATE is used to convert a partial century date (MMYYYY) into a keyword range suitable for a search on a PowerHouse internal date field.

```
%DATE(011990,MMYYYY,PHDATE)
```

Each of the %DATE expressions in the examples above is treated as a single keyword, and can be combined in any of the operations supported for keyword keys. In the example below, two %DATE expressions are combined in a range operation:

```
%DATE(09301990,MMDDYYYY,YYMMDD):%DATE(01011991,MMDDYYYY,YYM-
MDD)
```

Note that partial date entry-formats (like YYMM) are not supported for range operations.

Pattern matching

ODXFIND modes 3 and 5 support the following wildcard characters anywhere in a character keyword argument:

- ? represents any single printable character
- # represents any single digit (0-9) of an ASCII number
- @ represents any number of ASCII characters, including spaces

In relational expressions, wildcard tokens can only be used at the end of an argument string.

Modes 10 and 11

Modes 10 and 11 are used to qualify a list of indexed keyword values. This is referred to as a keyword-only search. Keyword-only searches are discussed under the "Using DataView" heading in the "Utilities" chapter of the *OMNIDEX ImagePlus SDK Administrator's Guide.*

In mode 10 or 11, the list must contain only one partial keyword (followed by an @), or a keyword range immediately followed by a semicolon (;) or space. This partial keyword or keyword range defines a list of keywords, each of which can be retrieved using ODXGETWORD.

ODXFIND mode 10 or 11 finds all the keywords that match a partial keyword value or fall within the range of two keyword values. These modes are used before retrieval of those keywords using ODXGETWORD. In mode 10, the number of qualifying keywords contained within the generic/range specification is also returned.

For example, DataView calls a mode 10 ODXFIND when a user enters an exclamation point (!) followed by a partial keyword or a keyword range at a prompt for a keyword key.

Mode 30

ODXFIND mode 30 lets you compress a qualified list of record numbers into a list of OMNIDEX SI values after a retrieval on a keyword key in a linked detail, which effectively converts individual records into record complexes. This is useful when continuing a search from a linked detail into other sets within the same domain.

The *keywords* parameter

The *keywords* parameter supports any of the operations discussed below. You can combine different operations in the same keyword list.

Boolean operations

ODXFIND supports the combining of arguments in Boolean operations. The operators and their operations are listed in Table 2-4, on the next page.

Internal operator	ODXFIND mode 3 or 5 operators	Boolean operation
, (Comma)	AND or a space	an intersection of records that contain the keywords it separates. A key must contain all keywords combined in an AND operation to qualify records.
+ (Plus sign)	OR	a union of records that contain the keywords it separates. A key can contain either keyword combined in an OR operation to qualify records.
- (Minus sign)	NOT	an exclusion of records that contain the keyword it precedes. Keys must not contain the keyword preceded by the NOT operator to qualify records. If NOT begins the keyword list of the first ODXFIND in succession, it qualifies only those records that do not satisfy the keyword arguments that follow it.
*	* or a leading AND or OR operator	loads the records qualified in the most recent keyword search into memory. An asterisk is typically used to progressively qualify a list of records. A leading AND or OR operator can precede additional search arguments, to reload and progressively qualify records (internal ID list).
N/A	( )	used to nest Boolean expressions, and override the precedence of operations, discussed next.

Table 2-4: ODXFIND Boolean operations and tokens

When you combine several keywords in several different operations (as in DISK OR DISC AND DRIVE NOT PC), the order of operations for all ODXFIND modes except 5 is (range or relational operation, then) OR, then NOT, then AND. For ODXFIND mode 5, the precedence is (range or relational operation, then) NOT, then AND, then OR.

To change the order of operations for a search, nest Boolean and range expressions in parentheses. For example, the following keyword list for an ODXFIND mode 3 call qualifies all records with either SOFTWARE AND DEVELOPMENT, OR APPLICATION AND PROGRAM@ in the name, but NOT records with CONSULT@.

```
(SOFTWARE AND DEVELOPMENT) OR (APPLICATION AND PROGRAM@) NOT CON-
SULT@
```

The ODXFIND order of operations (without parentheses) would qualify records with DEVELOPMENT OR APPLICATION, NOT CONSULTING, AND SOFTWARE AND PROGRAM@.

Ranges

A keyword range may be specified by entering a starting value, the range operator and an ending value. The default range operator is a colon (:). Either the starting value or the stopping value is optional.

The syntax is: [*startvalue*]:[*stopvalue*]

startvalue:*stopvalue*	means all keywords from *startvalue* through *stopvalue*.
startvalue:	means all keywords greater than or equal to *startvalue*.
:*stopvalue*	means all keywords less than or equal to *stopvalue*.

When mode 3 or 5 is used, the tokens TO and THRU can be used in place of the colon.

If the *startvalue* and *stopvalue* values are numbers and the field is a character field (TurboIMAGE types U, X and Z), they must contain the same number of digits. A numeric ordering is used to determine the list.

If either value is non-numeric (TurboIMAGE types U or X), the values need not contain the same number of characters, and an alphanumeric ordering is used. For example, ASCII values like 123, 12BA, 12/22, 123.5 and 12.3 fall within the alphanumeric range 110:130A, but only 123 and 123.5 fall within the numeric range 110:130.

ASCII ranges also can be generic (partially specified), by appending an at sign (@) to a partial *startvalue* or *stopvalue*.

ASCII ranges must have the same number of characters in the start and stop values. A range of 130:3000 would return the error message `Start and stop values must have the same number of digits`. The range could, however, be specified as 130:999+1000:3000.

Relational operations

You can perform relational operations either by calling ODXFIND in mode 3 or 5, or by combining open-ended range operations with Boolean operations.

In mode 3 or 5 calls, any of the following relational operations can be performed using one of the tokens listed. For all other modes, a range operation, combined with a Boolean operation, can be used to simulate a relational retrieval. Relational operations are summarized in Table 2-5:

Relational operation	Mode 3 and 5 Tokens	Mode 1 or 2 Range equivalent
equal to	=	*keyword*
greater than	>	*keyword:,-keyword*
greater than or equal to	>=	*keyword:*
less than	<	*:keyword,-keyword*
less than or equal to	<=	*:keyword*

Table 2-5: ODXFIND relational operations and tokens

Generic arguments

Keyword arguments also may be generic (partially specified), by appending an at sign (@) to a partial keyword. @ is a wildcard, so a generic keyword includes all the words or values that begin with the same characters. For example, MANAG@ includes MANAGE, MANAGER, MANAGING, etc.

NOTE

Generic values are not supported for searches on binary fields.

Soundex phonetic searches

To specify a phonetic, or Soundex, search on a designated Soundex key, append the Soundex operator (!) to the keyword(s) in question, for example ALAN! AND ANDERSON!. When used on any field other than a Soundex field, error message -224, `Not a Soundex field`, is returned.

If a Soundex field is grouped with other fields, be sure to specify the Soundex field name in the *field* parameter when calling ODXFIND for a Soundex retrieval.

External ID files

External ID files can be created by ODXTRANSFER mode 201. These files are used to select records using IDs qualified from one or more previous calls to ODXFIND in the same OMNIDEX domain. External ID files are often used to save an OMNIDEX search for future reference.

The IDs in the file are then passed through the ODXFIND *keywords* parameter by placing the file name, preceded by $, in the keyword list (like $*filename*). $*filename* can be used like and combined with other keyword arguments. ODXTRANSFER mode 201 is discussed on page 2-90.

When loading an ID file where two files with the same name exist, one permanent and one temporary, ODXFIND will load the permanent file and ignore the temporary.

Multifind

Multifind operations are used to link ODXFIND searches across OMNIDEX domains, even if the target domain is in another database.

In Multifind, values from records that qualified in a previous ODXFIND can be used to qualify records in a second, *target*, domain. The keyword values used in a Multifind search can be supplied either from memory, or from an external file.

For a Multifind search to use values supplied from memory, they must be SI values returned from a previous ODXFIND. When a Multifind is using SI values returned from a previous ODXFIND, an ampersand (&) in the *keywords* parameter is all that is required to supply the SI values from memory.

If you qualified records using the Record Specific key (a keyword key installed on a detail without the Record Complex option), be sure to compress the ID list before performing a Multifind from memory against a Record Complex key or a key in a master. To compress the ID list call ODXFIND mode 30.

Multifind can use search values that have been written to an external file. The file, created by ODXTRANSFER (see page 2-90), is passed through the *keywords* parameter preceded by an ampersand (&). For example, the *keywords* parameter would contain &*filename*, where *filename* represents the name of the external file containing the search values. For more information about creating files for Multifind searches, see the ODXTRANSFER section of this chapter.

Multifind requires that the target field in the target data set is a keyword key. The name of the target field for a Multifind search is passed via the *field* parameter, as in any ODXFIND.

The Multifind operator (&) must be specified as the first character in the *keywords* parameter. Any keywords or operations that follow a Multifind operation (& or &*filename*) are ignored.

When no records qualify

If a keyword in the keyword list does not exist for the specified OMNIDEX field or group, no entries qualify unless the non-existent keyword is combined with other keywords that do exist via an OR operation. Otherwise, error 217, `No entries contain <`*keyword*`>` is returned.

Even if all keywords in the list exist, no entries may qualify, based on the Boolean conditions specified in the list. For example, if two keyword arguments in an AND operation exist, but not in the same record or record complex, no entries qualify. Here, *status* word 1 is set to zero to indicate a successful call, while *status* words 12-13 returns a qualifying count of zero. This is consistent with a TurboIMAGE DBFIND, which succeeds if the specified search item exists, even if the chain count is zero.

This affects error checking in programs. Following an ODXFIND, your program must check for a nonzero value in *status* word one and perform error processing. It also must check for a qualifying count greater than zero. If the qualifying count equals zero, it should display a message saying `No entries qualify` and re-prompt for keywords. See the COBODXS.DEMO.DISC program for an example of error processing.

In batch applications, however, there is often no way to re-try a search if no records qualify. In fact, a reporting application could return erroneous results if a keyword search qualified no records, but the internal ID list was preserved. In most report applications, it is better to purge the internal ID list if a search qualifies no records. To purge the internal ID list, and avoid erroneous results in batch applications, use mode option 400 when calling ODXFIND to further qualify records. To use this mode option, add 400 hundred to the base mode value. For example, to perform a mode 5 ODXFIND without preserving the internal ID list, call ODXFIND in mode 405.

Back-out feature

After every call to ODXFIND in mode 1, 2, 3 or 5, except those that qualify no records, ODXFIND stores the list of currently qualified records in memory. This is called the *internal ID list*.

OMNIDEX maintains two internal ID lists:

- the ID list that resulted from the ODXFIND keyword search that you just performed
- the ID list, if one exists, that resulted from the ODXFIND keyword search that immediately preceded the search you just performed

This lets you back out of, or undo, the most recent ODXFIND keyword search. For example, suppose you qualified 100 CUSTOMER records, and you narrowed the list to 6 records by searching the STATE keyword key using the keyword **CA** (for California). If you then changed your mind, and decided you wanted to include records for the states of California, Washington and Arizona, you could back out to the previous internal ID list that reflected 100 CUSTOMER records. You could then retry your search of the STATE key using the argument **CA AND AZ AND WA**.

To back out of the most recent search, call ODXFIND in mode 1, 2, 3 or 5 and pass a less-than operator followed by a semicolon (<;) in the *keywords* parameter. Note that you can only back out of the last search you did. If you attempt several undo operations in succession, ODXFIND returns an error.

Reloading the internal ID list

Records from prior retrievals can be qualified by using either of two operations:

- For mode 1, start the keyword list with an asterisk (*).
- For mode 3 or 5, start the keyword list with a leading AND or OR operator.

After performing an ODXFIND, you can use an asterisk, or a leading AND or OR operator (mode 3 or 5) to reload the last internal ID list. You can then further qualify the list, add to it or subtract from it by specifying additional keywords in Boolean operations.

A leading AND operator (or *, for mode 1) and any keywords that follow it intersects the IDs qualified in the current search to the ID list. A leading OR operator (or *+ for mode 1) and any keywords that follow it adds the IDs qualified in the current search to the ID list. A leading AND NOT operator (or *- for mode 1) and any keywords that follow it removes the IDs qualified by the current search from the ID list.

To reload the ID list without further qualifying it, place an asterisk (*) alone in the keyword list. A single asterisk reloads the internal ID list regardless of the *mode* value used in the call to ODXFIND.

OMNIDEX condition word values

Exceptional conditions

201	Operation stopped by user
205	Limit of undo stack reached
213	Too many keywords in list
217	No entries contain keyword
230	Can't open ID file
313	Index file empty

Calling errors

-201	Keyword an excluded word
-202	Illegal ODXFIND mode
-203	Data set not an OMNIDEX set
-204	Illegal use of NOT operator
-205	Bad keyword list
-206	Illegal use of range operator
-207	Keywords must start with an alphanumeric character or decimal point
-208	Start and stop values must have same number of digits
-209	Bad item name specified
-210	Item not an OMNIDEX keyword field
-211	Only one generic keyword or one keyword range permitted
-212	List must contain a generic keyword or keyword range
-213	Samelist not allowed in index-only modes (10 and 11)
-214	Not an OMNIDEX database
-216	Generic keywords not allowed on binary fields
-217	Keyword contains a non-numeric character
-218	Missing keyword in list
-219	Unmatched quote
-220	Missing terminator (semicolon or blank)
-221	Start and stop values must have same sign
-222	Split retrievals, RS/RC or RC/RS, not allowed with Samelist OR
-223	Dset must be same as in previous RS retrieval
-224	Not a Soundex field
-226	Range start and stop values must be keywords
-227	Number exceeds maximum allowed for this field
-228	Negative values not allowed for unsigned fields
-231	Not an ODXTRANSFER ID file
-232	Multifind cannot reference an ODXTRANSFER ID file
-233	File record size exceeds max keyword size
-234	File name cannot exceed 36 characters

ODXGET

ODXGET (*base, mode, status, si-list, si-count*)

ODXGET moves one or more OMNIDEX SIs (search items) or record numbers into the *si-list* parameter in preparation for a DBGET.

Parameters

base is the name of the integer array used as the *base* parameter when opening the database. The first element of the array must contain the base ID returned by DBIOPEN. (See page 2-44 for more information about the base ID.)

mode is a single, 16-bit word integer (0-4) that indicates the type of retrieval and the pointer movement.

- 0 rewind. Resets the pointer. The pointer is reset to the beginning of the list if the next operation is a forward read or spacing (mode 1 or 3). The pointer is set at the end of the list if the next operation is a backward read or spacing (modes 2 or 4). If the previous ODXFIND was performed on a record specific field, *status* word 16 contains the value -1. Otherwise it contains zero.
- 1 forward read. Reads the next *si-count* SIs, or relative record numbers, into the *si-list* array, and updates the internal pointer.
- 2 backward read. Reads the previous *si-count* SIs or relative record numbers into the *si-list* array and updates the internal pointer.
- 3 space forward. Moves the pointer forward by the integer count specified in *si-count*. Does not return any SIs or relative record numbers.
- 4 space backward. Moves the pointer backward by the count specified in *si-count*. Does not return any SIs or relative record numbers.

Mode options

In addition, mode options are used to return the IDs of individual detail records after record specific retrievals. To use a mode option, add its value to the appropriate mode value. For example, to perform a forward read of record numbers after an ODXFIND on a key in a linked detail set, add mode option 10 to mode option 1 to get mode 11.

10	returns the relative record numbers of qualified records in linked detail sets only.
20	returns the search item and the relative record numbers of qualified records in linked detail sets only.

status is the name of an array of twenty-one 16-bit words used to return information about the success of a call. See Table 2-1, on page 2-7, for a list of error condition codes. If a call is successful the contents of *status* array are:

Word	Contents of *status* array
1	TurboIMAGE condition word is zero.
2-10	not used.
11	OMNIDEX condition word is zero.
12-13	number of SIs or record numbers returned in *si-list* for modes 1-4. Number of qualifying IDs for mode 0
14	search item length in bytes
15	current OMNIDEX data set number
16	contains a nonzero value if the previous ODXFIND was performed on a record specific (non-Record Complex) key in a detail set; otherwise contains zero.
17-21	not used

If *status* word 1 is +888, *status* word 11 contains the OMNIDEX condition word. Call DBIERROR or DBIEXPLAIN to interpret the error. See the "OMNIDEX Condition Word Values" section at the end of this intrinsic description for a list of error numbers and conditions.

si-list is the name of the array into which the requested SIs or record numbers are moved. The array must be defined as binary or ASCII, depending on the data type of SIs to be returned, and must be large enough to hold the number of SIs specified by *si-count*.

si-count is a single 16-bit word integer (between 1 and 4096) that tells how many SIs or record numbers to move into *si-list*. If *si-count* exceeds the number of IDs remaining in the ODXFIND internal ID list, all the remaining SIs are returned. If no IDs are left to be moved, ODXGET returns an error. *status* words 12-13 contain the number of SIs moved.

In mode 3 or 4, *si-count* specifies the number of positions the pointer is to be moved forward or backward through the internal ID list. This number can be between 1 and 32,767. If the requested *si-count* would move the pointer to a position before the beginning, or past the end, of the file, an error is returned and the pointer's position is left unchanged.

Discussion

The internal ID list

After a successful call to ODXFIND, a list of one or more OMNIDEX IDs is accumulated. This internal ID list represents qualified records or record complexes. The OMNIDEX IDs in the internal ID list can take one of three forms, as discussed below.

Search items

After an ODXFIND (keyword search) on a key in a master set, or on a key installed with the Record Complex (RC) option, or after an ODXFIND mode 30, the internal ID list consists of TurboIMAGE search items (SIs). In this case, do not use any mode options when calling ODXGET to retrieve the SIs. Allocate the *si-list* parameter to receive one or more SIs, as specified by the value in *si-count*.

Relative record numbers

If the ODXFIND (keyword search) was performed on a key in an unlinked (Detail Record indexed) detail set, the internal ID list consists of TurboIMAGE relative record numbers. In this case, do not use any mode options when calling ODXGET to retrieve the relative record numbers. Allocate the *si-list* parameter to receive one or more 32-bit integer record numbers, as specified by the value in *si-count*.

Search item/record number combinations

If the ODXFIND (keyword search) was performed on a record specific (non-Record Complex) key in a linked detail set, the internal ID list consists of search item / relative record number combinations that represent the individual detail records of detail chains. You can use ODXGET to retrieve either SIs, relative record numbers or a combination of the two, depending on the mode option you use. To use either mode option, add it to the base mode value, as discussed next.

To return relative record numbers, use mode option 10. Allocate the *si-list* parameter to receive one or more 32-bit integer record numbers, as specified by the value in *si-count*. For example, ODXGET mode 11 returns the next *n* relative record numbers from the internal ID list, where *n* is the value of *si-count*. ODXGET mode 13 moves the record pointer *n* positions forward in the internal ID list, where *n* is the value of *si-count*.

To return search item/relative record number combinations, use mode option 20. Allocate the *si-list* parameter[1] to receive one or more search item/relative record number combinations, as specified by the value in *si-count*. For example, ODXGET mode 21 returns the search item plus the relative record number for the next *n* detail records qualified in the search, where *n* is the value of *si-count*. ODXGET mode 24 moves the record pointer *n* positions backward in the internal ID list, where *n* is the value of *si-count*.

ODXGET pointer movement and record ID retrieval

Each time ODXGET is called, OMNIDEX moves an internal pointer through the list of qualifying IDs. This pointer can move forward or backward, or be reset to the beginning of the ID list. ODXGET can either move the pointer alone, or move the pointer and return SIs, relative record numbers, or both, to the calling application. The SI or record number values are then used to retrieve records using standard TurboIMAGE calls to DBGET for master records, or DBFIND plus DBGET for detail records.

Immediately after an ODXFIND, or after an ODXGET mode 0, the pointer is at the beginning and end of the ID list. This means you can do a forward read from the beginning of the list, or a backward read from the end of the list. If you try to read past the end of a list, an error is returned.

The following examples of ODXGET retrievals assume that you have just qualified 10 IDs (SIs) using ODXFIND against keyword keys on a master. They illustrate how ODXGET would retrieve SIs and/or place the pointer in the marked positions. In all of the examples below, point A represents the pointer position immediately after an ODXFIND, or after a mode 0 ODXGET. This is the beginning and end of the ID list. Returned SIs (like SI 1 in the example on the next page) are represented by a lighter box around them.

1. The size of *si-list* depends on the size of the SI. For example, if the detail's OMNIDEX's SI is an X14 field, allocate 18 bytes for each combination retrieved.

Immediately after ODXFIND, starting at point A, an ODXGET mode 1 with an *si-count* of 1 retrieves SI number 1 and sets the pointer at the next ID (point B):

^ ^
A **B**

Starting at point A, an ODXGET mode 1 with an *si-count* of 5 retrieves SIs 1-5 and sets the pointer at point C (the sixth ID):

^ ^
A **C**

Starting at point A, an ODXGET mode 2 with an *si-count* of 1 retrieves SI number 10 and sets the pointer at point D:

^ ^
A **D**

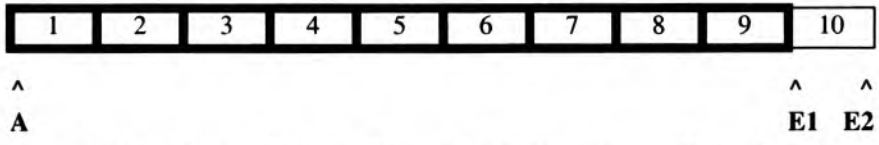
Starting at any point, an ODXGET mode 0 resets the pointer to point A, the beginning and end of the ID list:

^
A

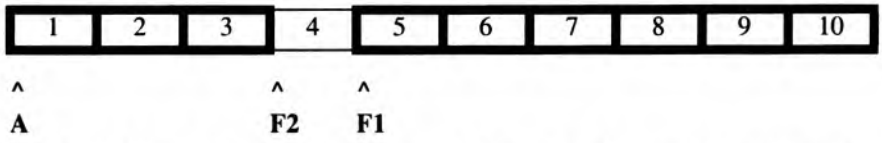
Starting at point A, an ODXGET mode 3 with an *si-count* of 10 would return an exceptional condition error, `End of ID List`, and would leave the pointer at point A:

^
A **end**

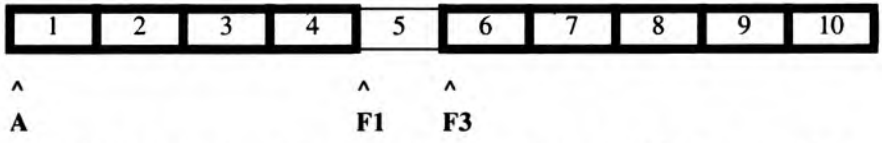
Starting at point A, an ODXGET mode 3 with an *si-count* of 9 would not retrieve any SIs and would move the pointer to point E1. A subsequent ODXGET mode 1 with an *si-count* of 1 would retrieve SI number 10 and would set the pointer at point E2 (the end of the list):



Immediately after an ODXFIND, an ODXGET mode 4 with an *si-count* of 6 returns no SIs and moves the pointer to point F1. Here, it moves backward by six positions in the list. A subsequent ODXGET mode 2 (backward read) with an *si-count* of 1 retrieves SI number 4, and sets the pointer at point F2:



If ODXGET mode 1 were used instead, SI number 5 would be retrieved and the pointer would be set at F3:



For OMNIDEX retrievals on unlinked detail sets, ODXGET returns the record number instead of the SI value. The record number is then used with DBGET mode 4 to perform the retrievals.

For OMNIDEX retrievals on record-specific (non-Record Complex) keys in a detail set, ODXGET mode option 10 or 20 may be added to modes 1 through 4. Mode option 10 returns the record number, instead of the SI value, for the qualifying detail record, which enables you to use a TurboIMAGE DBGET mode 4 to retrieve the record. Mode option 20 returns both the SI value and the record number.

OMNIDEX condition word values

Exceptional conditions

310 Beginning of ID list

311 End of ID list

313 Index file empty

314 End of range

396 OLCB damaged

397 Bad base ID, or database not opened using DBIOPEN

Calling errors

-300 Illegal mode

-301 Target array too small

-303 No preceding ODXFIND

-304 Illegal count

-305 Preceding ODXFIND not record specific

-314 Not an OMNIDEX database

ODXGETWORD

ODXGETWORD (*base, mode, status, target*)

ODXGETWORD reads the next keyword in the list qualified in the last mode 10 or 11 ODXFIND. This performs what is called a *keyword-only* retrieval.

Parameters

base is the name of the integer array used as the *base* parameter when opening the database. The first element of the array must contain the base ID returned by DBIOPEN. (See page 2-44 for more information about the base ID.)

mode is a single, 16-bit word integer (1 or 2) as follows:

1	Returns the next keyword in ascending sequential order.
2	Returns the next keyword in ascending sequence and the number of entries that contain the keyword.

status is the name of an array of 21 16-bit words used to return information about the success of a call. See Table 2-1, on page 2-7, for a list of error condition codes. If the call executed successfully, the contents of the array are:

Word	Contents of *status* array
1	Condition word is zero.
2-10	Not used.
11	OMNIDEX condition word is zero.
12-13	For mode 2, these words contain the number of entries that contain the specified keyword.
14-15	Double-word zero.
16-21	Not used.

target is the name of a 32 byte array into which the next keyword in the range is to be moved.

Discussion

Note that ODXGETWORD automatically converts binary field data into ASCII characters.

ODXFIND is first called using mode 10 or 11. The *keywords* parameter must contain one partially specified keyword or a keyword range.

ODXFIND mode 10 returns the number of qualifying keywords in the specified range and sets an internal pointer to the first word in the list. Mode 11 does not return a count, but sets the pointer to the first word in the list.

ODXGETWORD is then called to move one keyword into the target array. ODXGETWORD can be called repeatedly to retrieve all the keywords in the list. After the last qualifying keyword is reached, an `End of Range` condition is returned.

OMNIDEX condition word values

Exceptional conditions

314 End of Range

Calling errors

-300 Illegal mode specified

-302 No preceding mode 10/11 ODXFIND

ODXINFO

ODXINFO (*base, qualifier, mode, status, info*)

ODXINFO returns information about how keyword keys are installed on the database: what fields are indexed, how keys are grouped, and so on.

Parameters

base is the name of the integer array used as the *base* parameter when opening the database. The first element of the array must contain the base ID returned by DBIOPEN. (See DBIOPEN on page 2-44 for more information about the base ID.)

qualifier is the name of an eight 16-bit word (16 byte) array that contains a data set name or a single, 16-bit word integer data set number for which information is requested. In mode 6, the *qualifier* array also contains an integer group number in word 9 (byte 17).

mode is a single, 16-bit word integer value that specifies what type of information is desired. The modes, the type of *qualifier* necessary for each mode, and the information returned to the *info* parameter are shown in the tables that follow.

status is the name of an array of twenty-one 16-bit words used to return information about the success of a call. See Table 2-1, on page 2-7, for a list of error condition codes. Word 2 contains the number of words of information returned to the INFO area.

If the call executed successfully, *status* word 1 is zero. If *status* word 1 is +888, *status* word 11 contains the OMNIDEX condition word. Call DBIERROR or DBIEXPLAIN to interpret the error. See the OMNIDEX Condition Word Values section below for a list of error numbers and their corresponding error conditions.

info is an array in which information about OMNIDEX is returned. The format of this information depends on the *mode*. Refer back to the mode descriptions to see the contents of the *info* parameter.

mode value	Purpose	*qualifier* contents	*info* element	array contents	Comments
1	Identifies keyword keys for a set	OMNIDEX data set name (UPPER CASE) or number	1	data set number	
			2	set number of set's OMNIDEX master	
			3	item number of set's OMNIDEX SI	Elements 3 and 4 contain zero if the set is an unlinked detail
			4	offset (in words) of set's OMNIDEX SI	
			5	number of keyword keys in the set	
			6 . . . *n*	item number of a keyword key	This array must hold one 16-bit word for as many keyword keys as are counted in element 5.
2	Identifies keyword key groups for a set	OMNIDEX data set name (UPPER CASE) or number	1	data set number	
			2	set number of set's OMNIDEX master	
			3	item number of set's OMNIDEX SI	Elements 3 and 4 contain zero if the set is an unlinked detail
			4	offset (in words) of set's OMNIDEX SI	
			5	number of keyword keys in the set	
			6	item number of each keyword key in the set	Elements 6 and 7 each must hold one 16-bit word for as many keyword keys as are counted in element 5. The item number and group number for each keyword key in the set are returned in these two 16-bit words.
			7 . . . *n*	group number of each keyword key in the set (0 means key is ungrouped)	
3	Identifies all sets linked to the specified master	OMNIDEX master set name (UPPER CASE) or number	1	master's set number	
			2	number of sets in master's domain	This includes the master set specified in *qualifier*.
			3 . . *n*	set number of each set in domain	This element contains as many 16-bit words as there are sets counted in element 2.

Table 2-6: ODXINFO mode values

mode value	Purpose	*qualifier* contents	*info* element	array contents	Comments
4	Identifies all key groups in the domain of the specified master	OMNIDEX master set name (UPPER CASE) or number	1 2 3 . . *n* 4 . . *n*+1	master's set number number of groups in master's domain group number of a keyword key number of data set where keyword key occurs	Elements 3 and 4 each must contain as many 16-bit words as there are groups counted in element 2, up to a maximum of 63.
5	Identifies all OMNIDEX master sets (SI domains) for a database	(ignored)	1 2 . . *n*	total number of master sets data set number of each master set	This element contains as many 16-bit words as there are masters counted in element 1.
6	Identifies the keyword keys that belong to the specified group	OMNIDEX data set name (UPPER CASE) or number, plus a group number in word 9 (byte 17)	1 2 3 . . . 4 . . *n*	data set's number number of keyword keys in the group item number of a keyword key number of set where key resides	Must be an OMNIDEX master or unlinked detail. Elements 3 and 4 each must contain as many 16-bit words as there are keyword keys counted in element 2

Table 2-6: ODXINFO mode values

mode value	Purpose	*qualifier* contents	*info* element	array contents	Comments
7	Identifies all key options installed on the specified data set	OMNIDEX data set name (UPPER CASE) or number	1 2 3 4 5 6 . . *n* 7 . . *n*+1	data set's number data set number of set's OMNIDEX master item number of set's OMNIDEX SI offset (in words) of set's OMNIDEX SI number of keyword keys in the set item number of keyword key	Elements 6 and 7 each must contain as many 16-bit words as there are keyword keys counted in element 5. The bit mapping for element 7 is summarized below: Bit:Length Option 0:1 No Exclude 1:1 No Parse 2:1 Record Specific 3:1 No Translate 4:1 Soundex 5:1 No Deferred 6:1 Batch Indexing 7:1 Blob Indexing 8:4 (not used) 12:4 Type Cast as: 0 ASCII (U, X, Z) 1 Integer (I or J) 2 Logical (K) 3 Packed (P) 4 Real (R) 5 IEEE floating point (E) 6 Not used 7 Zoned (Z)
8	Indicates the type of OMNIDEX ID used for a master or unlinked detail set	OMNIDEX master or unlinked detail set name (UPPER CASE) or number	1	a numeric code for the type of ID installed	0 means SI/ID -1 means transparent ID -2 means DR indexed (unlinked) detail set

Table 2-6: ODXINFO mode values

mode value	Purpose	*qualifier* contents	*info* element	array contents	Comments
9	Indicates the next "free" ID value for the specified master	OMNIDEX master set name (UPPER CASE) or number	1	next "free" ID value for domain. May be an ID previously used by a deleted record	Contains one 32-bit word
10	Indicates the next "unused" ID value for the specified master	OMNIDEX master set name (UPPER CASE) or number	1	next "unused" ID value for the domain	Contains one 32-bit word
20	Indicates number of OMNIDEX indexes in base	none	1	index count (number of keyword fields and OMNIDEX composite keys)	
312	Indicates whether the specified keyword key is a composite key, and the location of composite key component	OMNIDEX data set name (UPPER CASE) or number, plus a keyword key's name or item number in word 9 (byte 17)	1 2 3 4 5 6	key's item number key's group number key's length (bytes) number of components in composite key or compound item record byte offset of component length of component	A number of 1025 or greater identifies the key as a composite For compound item, this value represents the subitem length. For example, for a 5X40 key, this value would be "40." If this number is negative it represents the number of elements of a compound array. For example, for a 5X40 key, this value would be "-5". Elements 5 and 6 each must contain as many 16-bit words as there are key components counted in element 4*.

Table 2-6: ODXINFO mode values

* If the number returned to element 4 is negative, indicating a compound item, elements 5 and 6 occur only once.

OMNIDEX condition word values

Calling errors

-500 Illegal mode

-501 Not an OMNIDEX data set

-502 Data set not an OMNIDEX master

-503 Undefined OMNIDEX group

-505 Not an OMNIDEX keyword field

-514 Not an OMNIDEX database

ODXPRINT

ODXPRINT (*filename, keywords, control, status, plabels*)

ODXPRINT prints the contents of an ASCII, EDIT/3000, QEDIT or TDP text file to the line printer. The formal file designator, ODXPRINT, is opened with shared lock access.

Parameters

filename is an array containing the name of the file to be printed. The file name can contain a lockword, group name and account name. It must be terminated with a space or a semicolon (;).

keywords is ignored by ODXPRINT.

control is a ten 16-bit word (20 byte) array. Words 3 and 8 through 10 are ignored. The remaining words contain the following:

Word	Contents of *control* parameter
1	contains 1 to suppress page headings, otherwise contains zero
2	the number of lines per page
3	not used
4-5	line number in the file where the printing should start. Files are assumed to have line numbers starting at 1, in increments of 1, whether they are numbered or not. For example, the 20th record in the file is treated as line 20, even if a different editor line number is assigned. A line number of zero means the first line of the file, which is the standard
6-7	line number in the file where printing should stop. This is typically set to zero. (Zero means the last line in the file.)
8-10	not used

status is the name of an array of twenty-one 16-bit words used to return information about the success of a call. If the call executed successfully, *status* word 1 is zero. If the call failed, *status* word 1 is +888 and *status* word 11 contains one of the error codes listed on the next page.

plabels is the name of a ten 16-bit word (20-byte) array that must be initialized to all zeros before the first call to ODXPRINT. After this call, the contents of the array must be preserved (not changed) for subsequent calls to ODXPRINT to function properly.

OMNIDEX condition word values

Calling errors

802 Unable to open the specified file

803 Unable to write to the specified file

804 Unable to read the specified file

805 Unable to access a session temporary file

ODXTRANSFER

ODXTRANSFER (*base, mode, status, filename, options*)

ODXTRANSFER copies to a file the SIs, IDs, or field values of records qualified in an ODXFIND search.

Parameters

base is the name of the integer array used as the *base* parameter when opening the database. The first element of the array must contain the base ID returned by DBIOPEN. (See DBIOPEN on page 2-44 for more information about the base ID.)

mode is a 16-bit word integer, either 1 or 2 for SIs, or 201 for IDs, as follows:

- 1 normal write mode for OMNIDEX SIs. Overwrites the contents of an existing file.
- 2 APPEND mode for OMNIDEX SIs. Appends the SIs to the end of an existing file. If the SIs are transferred to a new file, this parameter is ignored.

Mode options

In addition, there are several mode options. They are specified by adding the mode option value to the mode values 1 or 2 only.

For example, ODXTRANSFER mode 11 performs a normal write of record numbers after an ODXFIND on a non-RC key in a detail set.

- 10 transfers the record number instead of the search item (for non-Record Complex keys in linked details only.
- 20 transfers the combined search item and the record number (for non-Record Complex keys in linked details only).

Mode Options 10 and 20 are used for transfers of values returned from searches on record specific keys in detail sets, and can be added to modes 1 and 2 only.

- 100 transfers the data contained in the field specified in the *options* parameter to a file. You can use these files in a Multifind operation, as discussed on page 2-66.

200 ID file mode. Transfers IDs, from the internal ID list, to an external file. This file can then be passed in an ODXFIND keyword list by entering the file name preceded by a dollar sign (*$filename*), as discussed on page 2-66.

Note that OMNIDEX IDs cannot be transferred to a file previously created as an SI file because they do not have the same format and file code. Also, APPEND access is not permitted with OMNIDEX ID files.

status is the name of an array of twenty-one 16-bit words used to return information about the success of a call. If the call executed successfully, *status* word 1 is zero and *status* words 12-13 contain the number of SIs transferred. If *status* word 1 is +888, *status* word 11 contains the OMNIDEX condition word and words 12-13 are set to zero.

filename is the name of an array that contains the name of the file to which the SIs or IDs are to be transferred and is terminated by a blank.

options is the name an array containing sixteen 16-bit words (32-bytes) used only with mode option 100. The first 16 bytes contain the name of the specified OMNIDEX master or DR detail data set. If the data set name is less than 16 bytes (characters), it must be terminated by a space or semicolon. This may be a detail set in two cases:

- if the detail set is Detail Record indexed (DR) and therefore not linked to a master
- if the last ODXFIND was performed on a record-specific key. Qualified IDs must still be in RS form (not compressed to record complexes).

The second 16 bytes of the *options* parameter contain the name of the specified field (terminated by a space or semicolon) to be transferred from each qualified record in the data set.

Discussion

Modes 1 and 2

Use ODXTRANSFER mode 1 or 2 to transfer OMNIDEX SIs qualified by ODXFIND to a file. You can use this file to drive a report writer, or in a Multifind operation. ODXTRANSFER is very fast, because it uses MR/NOBUF writes for the transfer, and OMNIDEX SIs are retrieved from the OMNIDEX indexes.

The file specified by *filename* is created automatically by ODXTRANSFER. The file is created larger than is required by the current number of qualifying SIs to accept more SIs to be appended later. Still, no disk space is wasted, because unallocated file extents are used for the additional pad space.

You should not transfer data to an existing file, because ODXTRANSFER uses MR/NOBUF writes. Instead, let ODXTRANSFER create the file with the correct record size, capacity and blocking factor.

You can use a file equation to equate the formal file designator for a different file capacity. But never use a file equation to specify APPEND access. If you do, an error is returned. Specify mode 2 for APPEND access.

Mode 101

Use ODXTRANSFER mode 101 to transfer the contents of a specified field from records qualified by ODXFIND to a file for later use. Pass the field name and data set name through the *options* parameter. Typically, the files created by mode option 100 are used in Multifind operations by passing the file's name, preceded by an ampersand (&*filename*), through the *keywords* parameter of an ODXFIND call.

ODXTRANSFER mode 101 or 102 is much slower than other ODXTRANSFER operations, because ODXTRANSFER must retrieve each record from the OMNIDEX master or DR detail.

ODXTRANSFER mode 101 resets the item list to include only the field specified through the *options* parameter. Therefore, you must respecify the item list after a call to ODXTRANSFER mode 101 to reflect the items you wish to return in a TurboIMAGE call.

Mode 201

ODXTRANSFER mode 201 is used to transfer OMNIDEX IDs qualified by ODXFIND to a file for later use. This file of IDs can be reinstated by entering $*filename* in the keyword list for ODXFIND.

For example, if you did a long or complicated keyword retrieval that you needed to repeat later, you could save the qualified IDs to a file by calling ODXTRANSFER in mode 201. When you later needed to do that same retrieval, you could reference that file through the *keywords* parameter in a call to ODXFIND as $*filename*. This would reload that list of IDs.

The major factors in determining what ODXTRANSFER *mode* value to use are the type of domain on which the initial ODXFIND was performed, and how you will use the transferred information. Table 2-7 shows the different ODXTRANSFER *mode* values based on the type of domain where records were qualified, and the intended use of the transfer file.

Type of Domain	Data Set	*mode* value	Use of Transfer File
SI/ID or transparent	Master or Detail found by Record Complex keys.	1 or 2	&*filename* ODXFIND search across domains (Multifind)
SI/ID	Linked detail	11	reuse list of qualified record numbers via DBIGET mode 4
SI/ID	Linked detail	21	reuse list of qualified SIs to search other sets in domain
DR	Unlinked detail	101 or 102	&*filename* ODXFIND (Multifind) search across domains
Any	Master or detail	201	$*filename* ODXFIND operation to reload qualified IDs

Table 2-7ODXTRANSFER mode values compared

OMNIDEX condition word values

Calling errors

-800 Bad mode

-801 File record size must = OMNIDEX SI size

-802 File capacity too small

-803 Not an ODXTRANSFER ID file

-804 Insufficient stack space

-805 Can't transfer SIs to an ID file

-851 Need blank after filename

ODXVIEW

ODXVIEW (*filename, keywords, control, status, plabels*)

ODXVIEW displays the contents of an ASCII, EDIT/3000, QEDIT or TDP text file on the terminal screen.

Parameters

filename is an array containing the name of the file to be displayed on the screen. The file name can contain a lockword, group name and account name. It must be terminated with a space or a semicolon (;).

keywords is an array containing a list of keywords to be highlighted when displayed on the screen. Separate keywords with OMNIDEX Boolean operators and terminate the list with a semicolon (;) or space. A list of keywords for the ODXFIND *keywords* parameter is in the correct format. See the section on ODXFIND in this chapter for more information.

control is a ten 16-bit word (20 byte) array that contains the following:

Word	Contents of *control* parameter
1	highlight flag that controls the highlighting of words passed in the *keywords* array 0 = no highlighting 1 = HP terminal highlighting 3 = ANSI terminal highlighting
2	number of lines displayed on the first screen page. Version 3 can display 100 lines per screen page, to use the advanced display capabilities of newer terminals
3	number of lines, up to 100, displayed on subsequent screen pages
4-5	number of first line to display. Files are assumed to have line numbers starting at 1, by increments of 1, whether numbered or not. For example, the 20th record in the file is treated as line 20, even if a different editor line number is assigned. Zero or 1 displays the first line of the file. A negative value (-1, for example) means scan the file starting at the specified line (here, 1) until a word in the *keywords* array is found. Then back up half a page, and begin displaying the document.

6-7 number of last line to display (zero means the last line in the file)

8 containing a value of 1 enables the following command line:

```
(B)egin,(E)xit,(F)irst,(L)ast,(N)ext,(P)rev,
(R)ewind,(S)can,line#,[Return]:
```

9-10 not used

status is the name of an array of twenty-one 16-bit words used to return information about the success of a call. See Table 2-1, on page 2-7, for a list of error condition codes.

plabels is the name of a ten 16-bit word (20-byte) global array that must be initialized to all zeros before the first call to ODXVIEW. After this call, the contents of the array must be preserved (not changed) for subsequent calls to ODXVIEW to function properly.

Word 8 of *plabels* is used to specify an inactivity time-out in seconds. Values can range from 60 to 600 (seconds), or 0 (zero) which disables the time-out.

Discussion

ODXVIEW displays one screen page of text, then pauses until a key is pressed at the terminal. The number of lines in a screen page is determined by words 2 and 3 of the *control* parameter. The file is opened with shared lock access.

One-letter commands may be entered from the terminal during the execution of a program that has called ODXVIEW. They are:

B rewind to beginning of file and reset F command

E exit ODXVIEW

F go back to first screen page

L go to last screen page

N display next screen page.

P display previous screen page.

R rewind to first screen page

S scan forward to next keyword occurrence

1-9 display next 1 to 9 lines

[return] display next screen page

Any other character causes ODXVIEW to close the file and return to the calling program. After the last line of a file is displayed, ODXVIEW waits for the user to enter the next command.

OMNIDEX condition word values

Calling errors

If the call executed successfully, *status* word 1 is zero. If the call failed, *status* word 1 is +888 and *status* word 11 contains one of the following numbers:

802	Unable to open specified file
803	Unable to write to specified file
804	Unable to read specified file
805	Unable to access a session temporary file

Standard Interface to Third Party Indexing

Calling TurboIMAGE intrinsics

From a programmer's standpoint, using the Standard Interface to Third Party Indexing with OMNIDEX involves learning a few extra *mode* values for DBFIND and DBGET. These *mode* values invoke search and retrieval routines that use OMNIDEX indexes to find and get records.

How the interface works

The standard interface between TurboIMAGE and the OMNIDEX indexing product provides for automatic updating of OMNIDEX indexes, and the retrieval of records using OMNIDEX indexes through DBFIND and DBGET. The TurboIMAGE intrinsics (version C.04.03 or later) can now update and access the OMNIDEX indexes directly by calling OMNIDEX routines in XLOMNIDX.PUB.SYS. These OMNIDEX routines access or update the indexes, and then return control to the calling TurboIMAGE intrinsic.

In most cases, the flow of TurboIMAGE intrinsic calls is as follows:

1. The TurboIMAGE intrinsic performs its standard TurboIMAGE function.
2. The TurboIMAGE intrinsic determines if a call to OMNIDEX intrinsics is appropriate, and if so, calls the routine.
3. The OMNIDEX intrinsics, when called, act against the OMNIDEX indexes, and then return control to the calling TurboIMAGE intrinsic.
4. If the OMNIDEX intrinsics *status* indicates that it completed successfully, TurboIMAGE completes the intrinsic call by performing any logging and recovery steps. Unsuccessful calls to update intrinsics are backed out by the MPE XL Transaction Manager or TurboIMAGE's dynamic rollback.
5. The TurboIMAGE intrinsic returns control to the calling program.

Using this method, the TurboIMAGE intrinsics insure that changes to the TurboIMAGE database are synchronized with changes to OMNIDEX indexes, including transaction logging, dynamic rollback, and other recovery strategies. This means that no programming changes are necessary for your existing TurboIMAGE update applications.

Also using this method, the TurboIMAGE intrinsics take advantage of the retrieval capabilities of the OMNIDEX indexes. The intrinsics DBFIND and DBGET continue to provide the traditional methods of access x forward and backward serial reads, forward and backward chained reads, calculated (or hashed), and directed reads. Additionally, DBFIND and DBGET provide new methods of performing generic key searches, sorted sequential retrievals, and keyword or multi-key searches.

OMNIDEX searches

When performing OMNIDEX searches, the *argument* parameter supports search operators, in addition to search values. For example, in a DBFIND mode 12 on an OMNIDEX keyword key, the *argument* parameter could contain the following argument:

```
(DISC OR DISK) AND EAGLE;
```

In this example, "DISC", "DISK" and "EAGLE" are search values, as are normally passed through the *argument* parameter. "OR" and "AND" are Boolean search operators that establish a logical relationship between the search values. The parentheses indicate that the OR operation should take precedence over the AND operation. The semicolon (;) terminates the argument list. If the field in the *item* parameter is a keyword key installed on a PART-DESCRIPTION field, the mode value (12) and argument tell DBFIND to locate all records with the words DISC or DISK, and EAGLE in the PART-DESCRIPTION field.

Identifiers for all the records with the words DISC or DISK, and EAGLE in the PART-DESCRIPTION field are stored in memory, pending further qualification on other keys, or retrieval of the actual records by DBGET. This list of record identifiers is called the *internal ID list*.

When you use a partial key value in a sorted retrieval, record selection is determined by the number of leading characters matched against indexed key values. For example, a search on a COMPANY key using the argument A would retrieve records with key values that begin with "A" (like "AARDVARK," "ABC," "ACE," and "ACME"). The argument AC qualifies the records for "ACE" and "ACME," but not "AARDVARK" or "ABC" because their two leading characters do not match the partially specified argument value "AC."

The more characters you specify, the more precise the retrieval criteria, and the more selective the retrieval.

The key value returned by a sorted retrieval is the first key value that qualifies for the specified operation and argument. For the relational operations > and >=, the first value returned is the first matching key value in ascending sequence, or the lowest key value that qualifies. For the relational operations < and <=, the first value returned is the first key value in descending key sequence, or the highest key value that qualifies.

For example, an index for a key might contain several key values starting with "L," ranging from "LABEL" to "LUCKY." A partial key >= operation on the argument value "L" returns "LABEL," while a <= operation returns "LUCKY."

Similarly, a > operation using the argument value "L" returns the lowest key starting with "M," and a < operation using the argument value "L" returns the highest key starting with "K." Some possible returns of relational retrievals are listed below.

Argument Value	First Key Value Returned
<L	KUDOS
>=L	LABEL
<=L	LUCKY
>L	MAN

Note that you can express relational operations by supplying tokens (>, >=, <=, <) in the *argument* parameter, or by using an appropriate *mode* value (like 2*nn* to perform a "greater than" retrieval).

For more information about (OMNIDEX) keyword and sorted key access, see "DBIFIND", on page 2-23, "DBIGET", on page 2-28, and "ODXFIND", on page 2-54.

About TurboIMAGE intrinsics

DBPUT, DBDELETE, and DBUPDATE have been modified to call OMNIDEX indexing routines (when the indexes are enabled) in addition to their standard updating functions. While this insures that the OMNIDEX indexes for a given database are automatically updated, the changes are virtually transparent to programmers and your current TurboIMAGE applications.

This section discusses the extended modes, features, and error codes of the TurboIMAGE intrinsics that apply to OMNIDEX. For complete information about the TurboIMAGE intrinsics, consult the *TurboIMAGE/XL Database Management System Reference Manual.*

DBCONTROL

DBCONTROL (*base, qualifier, mode, status*)

DBCONTROL allows a process that has exclusive access to the open database to enable or disable the "output deferred" option. In OMNIDEX applications, it also enables or disables the ID list preservation feature when a keyword search qualifies no records.

Parameters

base is the same as for current TurboIMAGE. (See "DBOPEN" in the *TurboIMAGE/XL Database Management System Reference Manual* for more information about the base ID.)

qualifier is the same as for current TurboIMAGE

mode is a 16-bit word that supports the following integer values:

1-2	same as for current TurboIMAGE
800	in a keyword search, maintains the internal ID list if current search qualifies no records.
801	in a keyword search, purges the internal ID list if current search qualifies no records.

status is the array containing TurboIMAGE information about the procedure. This array consists of ten 16-bit words. These words are the same as for current TurboIMAGE. In addition to the standard TurboIMAGE calling errors and exceptional conditions, there are also OMNIDEX calling errors and exceptional conditions.

Discussion

When progressively qualifying records by searching several keyword keys, the internal ID list is preserved by default if a keyword search that references the internal ID list qualifies no records. This is called the *back out feature*. For example, you may have qualified 100 CUSTOMER records using the argument SOFTWARE against a COMPANY-NAME key. If you attempted to narrow down those customers by searching the STATE keyword key using the keyword CA, and no records qualified, the IDs of the 100 CUSTOMER records would remain in memory. You could attempt another retrieval using another key (like ZIP-CODE) or a different argument (like CO). This feature is useful for online applications, where users may try different sets of criteria to qualify a desired set of records.

In batch applications, however, there is often no way to re-try a search if no records qualify. In fact, a reporting application could return erroneous results if a keyword search qualified no records, but the internal ID list was preserved. In most report applications, it is better to purge the internal ID list if a search qualifies no records. To purge the internal ID list, and avoid erroneous results in batch applications, call DBCONTROL in mode 801 before using DBFIND to further qualify records. If, after qualifying records in batch, you want to re-enable the back out feature, call DBCONTROL in mode 800 before the next call to DBFIND.

DBFIND

DBFIND (*base, dset, mode, status, item, argument*)

DBFIND qualifies a chain of records from the set (or table) specified in the *dset* parameter. It does this by comparing the search value passed in *argument* against the values indexed for the key specified in the *item* parameter.

DBFIND also sets up pointers for TurboIMAGE chained access when used against a search item in a detail set.

The *argument* parameter can contain a full, partial, or pattern-matched search value that qualifies a chain through TurboIMAGE, sorted, or keyword key access. For mode 12 searches, the *argument* parameter can also contain Boolean, range, and relational operators. The *argument* parameter can also contain an SI value that locates a chain head through TurboIMAGE access.

When you qualify a chain, a count of the number of records in the chain (those whose key values meet the criteria supplied in the *argument* parameter) is returned to the *status* parameter.

Parameters

base is the array used as the *base* parameter when opening the database. The first element of the array must contain the base ID returned by DBOPEN. (See "DBOPEN" in the *TurboIMAGE/XL Database Management System Reference Manual* for more information about the base ID.)

dset is the array containing the data set to access. *Dset* is either a left-justified name or a 16-bit number. The data set name can be up to 16 characters long. If shorter than 16 characters, the name must be terminated by a semicolon (;) or space (for example, CUSTOMERS; or ORDER-LINES).

mode is a 16-bit word that accepts the following integer values:

1 performs a TurboIMAGE DBFIND, if *item* is a TurboIMAGE search item on a detail data set.

When the IMAGETPI JCW is set to 400, and the *argument* parameter contains wildcard characters (#, ?, @), or any OMNIDEX-supported operators (such as parentheses, AND, *, or >), or is terminated by a semicolon or trailing blanks, an OMNIDEX search is performed, a record pointer is set, and a chain count is returned. This is discussed later, in "Argument terminators", on page 2-108.

Wildcard characters are not supported for binary keys. Use DBFIND mode 11 for range searches against binary sorted keys.

10 performs a standard TurboIMAGE DBFIND on a detail search item passed through the *item* parameter

11 performs a range retrieval for numeric sorted keys of types E, I, J, K, P, R, and Z. *Argument* contains the start and stop values for the range, which must be passed as the same data type and length as the key. You can use an open-ended range by supplying a maximum key value for the stop value, or minimum key value for the start value.

12 performs a keyword search and returns a count of the number of records or record complexes that qualified

To perform a keyword search, which qualifies records based on the contents of several keys, perform successive DBFINDs using mode 12.

See "Keyword searches", on page 2-112, for more information about the operations supported for a mode 12 DBFIND.

13 undoes the last mode 12 or 23 OMNIDEX keyword search, and restores the previously qualified list of records (internal ID list)

21 same as mode 1 in function and *argument* syntax, but does not return a chain count

22 same as mode 11 in function and *argument* syntax, but does not return a chain count

23 same as mode 12 in function and *argument* syntax. Returns the number of record complexes when used against Record Complex keys (see page 2-112).

Sorted relational positioning modes

You can use relational positioning modes to perform a relational operation using the search value passed through *argument*. These *mode* values do not return a chain count. They simply position the current record pointer at the appropriate place in the sorted key sequence. A condition of "17" is returned to the *status* array when no records satisfy the search value in *argument*.

The *mode* parameter uses *nn* to represent the length of a partial key argument. A positive *mode* value means that *nn* is the argument 16-bit word length. A negative *mode* value means that *nn* is the argument byte length.

1*nn* performs an "equal to" (=) search, and sets the record pointer before the first record that matches the first *nn* words or bytes of the partial value in *argument*. A *mode* value of 100 sets the pointer to the record with the lowest key value and qualifies all entries in the data set. A *mode* value of 199 uses the value in *argument* as the full key value.

2*nn* performs a "greater than" (>) search, and sets the record pointer before the first record greater than the first *nn* words or bytes of the partial value in *argument*. A *mode* value of 200 sets the pointer to the record with the lowest key value and qualifies all entries in the data set. A *mode* value of 299 uses the value in *argument* as the full key value.

3*nn* performs a "greater than or equal to" (>=) search, and sets the record pointer before the first record that either matches or is greater than the first *nn* words or bytes of the partial value in *argument*. A *mode* value of 300 sets the pointer to the record with the lowest key value and qualifies all entries in the data set. A *mode* value of 399 uses the value in *argument* as the full key value.

4*nn* performs a "less than" (<) search, and sets the record pointer after the first record less than the first *nn* words or bytes of the partial value in *argument*. A *mode* value of 400 sets the pointer to the record with the highest key value and qualifies all entries in the data set. A *mode* value of 499 uses the value in *argument* as the full key value.

5*nn* performs a "less than or equal to" (<=) search, and sets the record pointer after the first record that either matches or is less than the first *nn* words or bytes of the partial value in *argument*. A *mode* value of 500 sets the pointer to the record with the highest key value and qualifies all entries in the data set. A *mode* value of 599 uses the value in *argument* as the full key value.

status is the array containing TurboIMAGE information about the procedure. This array consists of ten 16-bit words. These words are listed below:

Word	Contents of *status* array
1	the condition word. If the procedure succeeds, the return status is zero. If the procedure fails, an error condition (such as "17") is returned to element 1.
2	value of zero
3-4	current record number set to zero
5-6	count of number of entries in chain (chain count)
7-8	record number of last entry in chain (TurboIMAGE keyed reads only)
9-10	record number of first entry in chain (TurboIMAGE keyed reads only)

item is an array of eight 16-bit words that contains the left-justified name of either a sorted key or a TurboIMAGE detail data set search item

argument contains a search value (either full or generic), with optional operators. The search values can be sorted key values (for sorted keys), keyword values (for keyword keys), or TurboIMAGE search items (for TurboIMAGE SIs in detail data sets). A sorted search argument can be:

- a single value that can contain wildcards or begin with a relational operator
- two values that establish a range

A keyword search argument can be:

- one keyword, optionally containing wildcards or a relational operator
- two or more keywords separated by Boolean operators or range operators
- a leading Boolean operator followed by either 1 or 2 above

A semicolon (;) terminates the *argument* list for mode 1, 12, 21, or 23 DBFIND calls.

The data type of the value in *argument* depends on the *mode* value. The table below shows the expected data type for each *mode* value.

Mode	Key type	Chain count	Begin/End of Chain	Argument type*	Wildcards allowed**	Description
1	TurboIMAGE	Y	Y	Any	N	
1	keyword	Y	Y	ASCII	Y	
1	sorted	Y	Y	Any	Y	ASCII needs semicolon (;)
10	TurboIMAGE	Y	Y	Any	n/a	TurboIMAGE only
11	sorted	Y	Y	Any	N	Full start/stop values
12	keyword	Y	Y	ASCII	Y	keyword only; needs semicolon (;)
13	keyword	n/a	n/a	Ignored	n/a	Reset last mode 12 or 23 search
21	keyword	N	Y	ASCII	Y	Same as mode 1 but no chain count
21	sorted	N	Y	ASCII	Y	Same as mode 1 but no chain count
22	sorted	N	Y	Any	N	Same as mode 11 but no chain count
23	keyword	N	Y	ASCII	Y	Same as mode 12 but no chain count
1*nn*	sorted	N	N	Any	n/a	Equal to
2*nn*	sorted	N	N	Any	n/a	Greater than
3*nn*	sorted	N	N	Any	n/a	Greater than or equal to
4*nn*	sorted	N	N	Any	n/a	Less than
5*nn*	sorted	N	N	Any	n/a	Less than or equal to

Table 2-8: DBFIND mode and argument data types

* "Any" means that the type of argument matches the type of the key field.

** Arguments with wildcard characters are *not* allowed binary keys.

A sorted composite key is binary if any of its components are binary. A sorted composite key is ASCII only if all of its components are ASCII.

Discussion

This section discusses the following topics:

- searches performed in mode 1
- tokens supported in sorted searches
- binary range retrievals
- relational positioning modes
- returning the chain count
- keyword searches
- calling errors and exceptional conditions

Searches performed in mode 1 (or 21)

DBFIND mode 1 was enhanced to support sorted and keyword searches on any data set, in addition to finding chain heads in detail sets. This lets you perform OMNIDEX searches using your existing TurboIMAGE retrieval applications. When developing new applications, however, we recommend that you use the specialized OMNIDEX and TurboIMAGE access modes listed on page 2-105. This minimizes the processing required for searches, and prevents ambiguities that result when *item* refers to an item with different types of keys installed on it (such as TurboIMAGE and OMNIDEX sorted).

Whether a mode 1 call to DBFIND against a search item in a detail data set performs a standard TurboIMAGE search or an OMNIDEX search depends on whether or not the IMAGETPI job control word (JCW) is set to 400.

By default, calling DBFIND mode 1 performs a standard TurboIMAGE search for a chain head against the SI of a detail set. To enable OMNIDEX parsing of the *argument* parameter, and OMNIDEX searches, through DBFIND mode 1, you must set the IMAGETPI job control word to 400:

```
SETJCW IMAGETPI=400
```

Having to set this JCW prevents special characters (like , and -) in the *argument* parameter from automatically being parsed as OMNIDEX operators.

To disable OMNIDEX parsing of the *argument* parameter, and OMNIDEX searches, through DBFIND mode 1, you must set the IMAGETPI job control word to zero:

```
SETJCW IMAGETPI=0
```

When IMAGETPI is set to 400, the kind of search that DBFIND does is determined by these factors:

- the type of keys (keyword, sorted, and/or TurboIMAGE SI) installed on the field referenced in *item*
- the data type of the key referenced in *item*
- whether the *argument* string is terminated
- whether the *argument* contains any operators or tokens supported by OMNIDEX (like AND, *, or >=)

Each of these factors is discussed next.

Keys referenced by the *item* parameter

The *item* parameter of DBFIND must refer to a keyed item. This item can be the SI of a detail set, a keyword key, a sorted key, or any combination of these three. If *item* refers only to an SI of a detail set, DBFIND uses standard TurboIMAGE access to locate records. If the *item* refers only to a sorted key, DBFIND uses sorted access to find records. If *item* refers only to a keyword key, DBFIND uses keyword access to find records.

Likewise, the key you reference in *item* must support the search values and operators you use in the *argument* parameter. For example, you cannot use Boolean operators in *argument* if the *item* parameter doesn't reference a keyword key. If the operations and search values in *argument* are not supported by the key referenced in *item*, DBFIND may return unexpected results.

Often, *item* may reference an item that is both a TurboIMAGE SI and an OMNIDEX key. When this happens, an ambiguity results. Take, for example, a detail table's SI that is also installed as both a sorted and keyword key. DBFIND must determine whether to perform a TurboIMAGE read, a sorted search, or a keyword search. To determine what type of search to perform, DBFIND checks the data type of the field referenced in *item*, and examines the *argument*.

The data type of the key in *item*

When *item* refers to an item that is keyed for several types of access (such as chained, sorted, and keyword), DBFIND mode 1 uses the data type of the item to help determine what kind of access to use.

If *item* refers to a binary SI (type I, J, K, R, or E) in a detail data set, DBFIND uses TurboIMAGE access to find records. If *item* refers to a character key (type U or X) the type of search performed depends on the *argument* used. If the *argument* contains no operators, tokens, or a terminator, then DBFIND uses it as a TurboIMAGE key value to locate a chain head. If the *argument* contains a terminator, tokens, or operators, DBFIND must then determine what type of OMNIDEX access (sorted or keyword) to use based on the types of operators and tokens used.

Argument terminators

When you end the *argument* string with a terminator, it helps DBFIND determine what kind of search to perform.

- When *item* refers to an ambiguous key (TurboIMAGE? Sorted? Keyword?), terminating *argument* tells DBFIND to use OMNIDEX access (keyword if Boolean operators are present, sorted if they are not).
- When *item* refers to a sorted key, a terminator tells DBFIND whether *argument* represents a full or partial-key (generic) value.

There are two ways to terminate the *argument* parameter:

- with a semicolon (;)
- with trailing blanks up to the length of the key

For TurboIMAGE DBFIND searches, don't use a terminator. For keyword searches, terminate the *argument* with a semicolon (;). For sorted searches, terminating *argument* with either a semicolon or trailing blanks indicates a full key value. Terminate the *argument* value with an at sign (@) to indicate a partial key value.

If the *argument* parameter does not contain a terminator, DBFIND terminates the argument based on the type of key being searched, as follows:

SI on detail set	DBFIND assumes that *argument* contains a full key value and performs a TurboIMAGE search.
sorted key	DBFIND assumes that *argument* contains a full key value and pads the argument with trailing blanks up to the length of the key.
keyword key	DBFIND uses the first two consecutive blanks as a terminator, or truncates the argument to the maximum length of the *argument* parameter (1024 bytes).

When *item* refers to the SI of a detail that is also an OMNIDEX key, and no terminator is used, DBFIND uses TurboIMAGE access to locate chain heads. If *item* refers to an item that is a sorted key, and a keyword key, DBFIND uses the presence of search operators to determine whether to perform a sorted search or a keyword search, as discussed next.

How the *argument* string determines access

When *item* refers to the SI of a detail that is also an OMNIDEX key, and the *argument* parameter contains wildcard tokens, search operators, or a terminator, DBFIND uses OMNIDEX access to find records. If *item* refers to an item that is installed with both keyword and sorted access, DBFIND determines which type of access to use based on the tokens present in *argument*.

If DBFIND detects any parentheses in the *argument* parameter, keyword access is used. If parenthesis are absent, DBFIND uses sorted access. Therefore, range (TO, :) and relational operators (<, <=, =, >=, >) elicit sorted searches in DBFIND mode 1 in the absence of parenthesis.

Tokens supported in sorted searches

DBFIND mode 1 searches on ASCII sorted keys support wildcard tokens. Wildcard tokens provide methods of searching for data without knowing all of the characters of a key. The wildcard tokens are:

- ? represents any single printable character
- # represents any single digit (0-9) of a number
- @ represents any number of ASCII characters, including spaces

Unless the argument includes a relational expression, you can place wildcard tokens anywhere in an argument. The @ wildcard must appear at the end of an argument when it is used with a ? or # token, or if it is used in a relational expression.

You cannot use wildcard tokens in searches against binary keys.

When an argument alone is supplied, an "equal to" (=) operation is assumed. However, you can include relational expressions in the *argument* parameter for mode 1 (or 21) searches on ASCII keys. This lets you perform sorted relational searches. The relational operators are:

- > greater than
- >= greater than or equal to
- < less than
- <= less than or equal to

For example, to find all records with a key value greater than "ABC", you would supply an *argument* value of >ABC.

You can use more than one relational operator in an *argument* (instead of representing a range). For example, the range "5 to 10" could be represented as ">=05<=10." Because the same number of digits must be supplied for both the start and stop values in an ASCII range, the leading "0" in ">=05" is required. When a wildcard is used in a relational expression, it must be at the end of the expression. In other words, the expression "<ABC@" is allowed but the expression "<ABC@DEF" is not allowed.

Although you cannot use relational operators on binary keys, they can be simulated by using open-ended binary ranges, as discussed next.

Binary range retrievals

Mode 11 supports ranges on binary keys. *argument* must be large enough to hold two full key values. Therefore, if the key referenced in the *item* parameter is a J2 key, then *argument* should be four words long. The start value of the range is specified in the first half of *argument*, and the stop value is specified in the second half. For example, to find records with key values that fall between "123" and "1456," you would supply a binary "123" in the first two words of *argument*, and a binary "1456" in the last two words.

You can use this technique to simulate >= and <= relational operations. To simulate a >= operation, supply the start value in the first part of *argument*, and the highest possible value in the second part (like binary 2,147,483,647 for a J2 field). To simulate a <= operation, supply the lowest possible value in the first part of *argument* (like binary zero), and the stop value in the second part. For example, to find all records with key values less than or equal to 123, enter 0 for the first part of *argument*, and a binary 123 for the second part.

Relational positioning modes

Modes 1*nn* through 5*nn* support searches on either ASCII or binary keys using a single argument of the same data type as the key (*item*) being searched. These modes do not support wildcard tokens or relational operators. Modes 1*nn* through 5*nn* simply position a record pointer in anticipation of a DBGET chained read.

The *mode* value must express the relational operation used to set the pointer (like "3" for "greater than or equal to"), and the length of *argument* (as *nn*). If the length expressed by *nn* is in words, the entire *mode* value is expressed as a positive integer. For example, a >= retrieval using a partial key length of 4 words is expressed as a *mode* value of 304. If the length expressed by *nn* is in bytes, the entire *mode* value is expressed as a negative integer. For example, a >= retrieval using a partial key length of 8 bytes is expressed as a *mode* value of -308.

When using partial key lengths of 100 bytes or more, express the partial key length in words. If you do not, the resulting *mode* value would signify a different relational operation and *argument* length. For example, a >= retrieval using a partial key length of 108 bytes (54 words) is expressed as a *mode* value of 354, not -408, which expresses a < retrieval using a partial key length of 8 bytes.

Returning the chain count

Modes 1 and 11 cause IMSAM to count all keys that satisfy the criteria expressed in *argument*. Although this happens very quickly, retrievals that qualify many key values may cause DBFIND to hesitate while it counts the key values. If you do not need a count of the records that qualified for a retrieval, consider using modes 21 and 22, instead of modes 1 and 11.

Keyword searches

Modes 12 and 23 support keyword searches on OMNIDEX keys. Keyword searches can use wildcard characters. They can also use relational, range, and Boolean operations to establish a relation between the search value in *argument* and the records being sought.

If Boolean operators are present in the *argument* parameter of a mode 1 call to DBFIND, and the *item* references a keyword key, a keyword search is performed automatically.

Boolean operations are used to establish a relation among several search values using a given OMNIDEX key. When the search involves several keys in successive calls to DBFIND, they can establish a relation between the current list of qualified records and the records that qualified for a previous search.

Keyword searches using DBFIND are very similar to keyword searches using ODXFIND. The primary difference is that the order of operations for DBFIND (NOT, AND, OR) is different than the default order (OR, NOT, AND) of precedence for ODXFIND (modes 1, 2, and 3). ODXFIND mode 5 uses the same order of operations as DBFIND.

Record Complex keys

When performing keyword searches on Record Complex keys, remember that entire chains qualify, not individual detail records. The qualifying count returned reflects the combined number of details in the chains that qualified.

Progressively qualifying detail records using Record Complex keys can produce misleading results. Because Record Complex keys qualify record complexes, and therefore entire detail chains, it is possible to qualify chains where no single record contains the keywords specified for the keys searched. Take, for example, the TICKLER and INITIALS Record Complex keys installed on a CUSTOMER-NOTES detail set. If you supply an argument of WW against the TICKLER key, and ADG against the INITIALS key, you'll qualify some CUSTOMER-NOTES chains where one record contains ADG in its INITIALS field, and another record contains "WW" in its TICKLER field.

If you must qualify individual detail records, have your data administrator install non-Record Complex keys on that detail data set. You can install composite keyword keys that duplicate items in a detail to afford both types of access to that detail. For example, you could install TICKLER and INITIALS as default keyword keys, to qualify individual detail records. You could also create TICKLERRC and INITIALSRC Record Complex composite keys to qualify detail chains.

Range operations

Range operations enable you to qualify a subset of records based on a range of search argument values. For example, the range operation BOB:CARL would qualify all records with keyword values between and including "BOB" and "CARL" (for example, "BOB", "BRIAN", "CALVIN" and "CARL").

The range operators are : (a colon) and TO. The syntax for a range operation is:

startvalue:*stopvalue*

or

startvalue TO *stopvalue*

Ranges on ASCII keys can be performed using generic search arguments as well. The range operation A@:C@ would qualify all records with keyword values between and including "AARON" and "CYRUS" (for example, "AARON", "BOB", "CALVIN", "CLEM" and "CYRUS").

Ranges can be open-ended, where one search argument is either preceded or followed by the range operator. For example, the range operation BOB THRU would return all records with keyword values between and including "BOB" and "ZEKE". Similarly, the range operation TO BOB would return all records with keyword values between and including "AARON" and "BOB".

Boolean operations

The Boolean operations and their tokens are shown in Table 2-9.

Tokens	Boolean operation
, AND	An AND operator between search values qualifies records that contain all of those values. The argument COMPUTER AND SOFTWARE or COMPUTER,SOFTWARE qualifies records with both "COMPUTER" and "SOFTWARE" in that keyed field. In successive calls to DBFIND, an AND operator at the beginning of an *argument* list intersects records that satisfy the argument with any previously qualified records. An AND operation is implied when two values are separated by a single space. For example, the argument COMPUTER SOFTWARE is equivalent to COMPUTER AND SOFTWARE.
+ OR	An OR operator between search values qualifies a union of records that contain either value. The argument COMPUTER OR SOFTWARE or COMPUTER+SOFTWARE qualifies records with either "COMPUTER" or "SOFTWARE" in that keyed field. In successive calls to DBFIND, an OR operator at the beginning of an *argument* adds records that satisfy the argument to any previously qualified records.
- NOT	A NOT operator between search arguments qualifies records that contain the first argument, but not the second. COMPUTER NOT SOFTWARE or COMPUTER AND NOT SOFTWARE, or COMPUTER,-SOFTWARE qualifies records with "COMPUTER" in that keyed field, but excludes those with "SOFTWARE" in that keyword field. A NOT operator at the beginning of an *argument* list qualifies records that do not satisfy the argument supplied. For example, NOT SOFTWARE qualifies records without SOFTWARE in that keyed field.
*	An asterisk (*) loads the records qualified in the most recent search into memory. Use it after calling DBGET to reload the same list of records as previously qualified.
( )	Parentheses are used to nest Boolean expressions to override the precedence of operations (NOT, AND, OR).

Table 2-9: DBFIND Boolean tokens and operations

Relational operations

Relational operations let you qualify records with keyword values equal to, greater than, greater than or equal to, less than, or less than or equal to a search argument. The relational operations and their corresponding tokens are shown below.

Tokens	Operation
= (default)	equal to
>	greater than
>=	greater than or equal to
<	less than
<=	less than or equal to

Table 2-10: DBFIND relational tokens and operations

To perform a relational search, precede the argument with a relational operator. In the expression <8912@, "<"is the token and "8912@" is the argument.

Calling errors and exceptional conditions

This section lists the calling errors and exceptional conditions for both TurboIMAGE and IMSAM.

TurboIMAGE exceptional conditions

17 No entry found matching argument

OMNIDEX calling errors

-3200 Illegal mode specified

-3201 Data set not an IMSAM detail

-3202 Key value exceeds defined key length

-3204 Item is not an IMSAM key

OMNIDEX exceptional conditions

3213 IMSAM tree is empty

-3029 START > STOP value

-3220 Missing terminator in Avg. No";"(semicolon)

-3025 Invalid key

DBGET

DBGET (*base, dset, mode, status, list, buffer, argument*)

DBGET returns records from the chain or internal ID list qualified by DBFIND.

Parameters

base is the array used as the *base* parameter when opening the database. The first element of the array must contain the base ID returned by DBOPEN. (See "DBOPEN" in the *TurboIMAGE/XL Database Management System Reference Manual* for more information about the base ID.)

dset is the array containing the accessed data set's left-justified name or 16-bit number. The data set name can be up to 16 characters long. If it is shorter than 16 characters, the name must be terminated by a semicolon (;) or a space (for example, CUSTOMERS; or ORDER-LINES).

mode is a 16-bit word integer. The TurboIMAGE and IMSAM modes follow:

TurboIMAGE modes

1 standard re-read

2 standard serial read

3 standard backward serial read

4 standard directed read

5 a forward read after a DBFIND. If the DBFIND used TurboIMAGE access, DBGET reads the detail chain. If the DBFIND used sorted access, DBGET retrieves records in sorted order from the indexes. If DBFIND used keyword access, DBGET retrieves records based on the identifiers in the internal ID list. When DBGET reaches the end of a chain or ID list, it returns a status condition of "15." See DBFIND for more information.

6 a backward read after a DBFIND (see mode 5, above). When DBGET reaches the beginning of a chain or ID list, it returns a status condition of "14".

7 standard calculated read

8 standard primary calculated read

11 resets pointer to the beginning of the record list qualified in a sorted search.

12 moves pointer forward by *n* records in the qualified records list for a sorted search, without actually retrieving any records. *n* is a 32-bit integer set in the *argument*.

13 moves pointer backward by *n* records in the qualified records list for a sorted search, without actually retrieving any records. *n* is a 32-bit integer set in the *argument*.

15 gets the next record in sorted order regardless of the search *argument* used to qualify the chain of records. DBGET mode 15 ignores the end-of-chain boundary as well.

16 gets the previous record in sorted order regardless of the search argument used to qualify the chain of records. DBGET mode 16 ignores the beginning of the chain boundary as well.

21 resets pointer to the beginning of the list of records qualified by keyword search (DBFIND mode 12 or 23).

22 moves pointer forward *n* entries in the list of records qualified by a keyword search, without actually retrieving a record. *n* is a 32-bit integer passed in *argument*.

23 moves pointer backward *n* entries in the list of records qualified by a keyword search, without actually retrieving a record. *n* is a 32-bit integer passed in *argument*.

25 retrieves the next record buffer from those qualified by a keyword search.

26 retrieves the previous record buffer from those qualified by a keyword search.

status is the array containing information about the TurboIMAGE procedure. This array consists of ten 16-bit words. These words are listed below:

Word	Contents of *status* array
1	the condition word. If the procedure succeeds, the return status is zero. If the procedure fails, an error condition is returned to element 1.
2	length of the logical entry read into the *buffer* array in halfwords
3-4	record number of the data entry read
5-6	zero for everything except a primary entry. A primary entry's element contains the number of entries in the synonym chain
7-8	record number of the preceding entry in the chain of the current path (TurboIMAGE keyed reads only)
9-10	record number of the next entry in the chain of the current path (TurboIMAGE keyed reads only)

list is the name of an array containing either an ordered set of data item names or numbers, an at sign (@) to specify all items in the data set, or an asterisk (*) to respecify items passed in the last *list*.

See the *TurboIMAGE/XL Database Management System Reference Manual* discussion of "DBGET" for more information about the *list* parameter.

buffer is the array where the values of data items specified in the *list* array are returned.

argument contains the full key value used to locate the record for modes 4, 7, and 8.

For modes 12, 13, 22, and 23 *argument* contains a 32-bit binary integer specifying the number of records to advance the pointer in the chain.

Discussion

DBGET reads through the chain of records qualified by DBFIND, and retrieves records through standard TurboIMAGE access. See the *TurboIMAGE/XL Database Management System Reference Manual* for more information about TurboIMAGE access. The IMSAM DBGET modes are discussed next.

Mode values for sorted retrievals

Modes 5 and 6 read forward and backward through a chain of records qualified by DBFIND (mode 1, 10, 11, 21, or 22). When a DBFIND is performed on a sorted key, record chains are always sorted from lowest key value to highest key value. To read or list records in ascending order (lowest to highest), use mode 5. To read or list records in descending order (highest to lowest), use mode 6.

Mode 11 resets the pointer to the beginning of a chain of qualified records.

Modes 12 and 13 let you skip forward or backward, by any 32-bit number of records, in a chain of qualified records. The number of records the pointer moves is passed through the *argument* parameter.

For example, if a DBFIND qualified a chain of 20 records, and you wanted to read the fifth record in the chain, you would perform a DBGET mode 12, with an *argument* value of 5 to move the pointer to the fifth record. Then, you would perform a DBGET mode 1 to read the fifth record. DBGET mode 13 serves the same function, but moves the pointer backward through a chain.

Immediately after a DBFIND mode 1*nn*, 2*nn*, or 3*nn*, you could use a mode 12 DBGET to skip ahead in the list. A mode 13 DBGET returns condition 14 (beginning of chain). Immediately after a DBFIND mode 4*nn*, or 5*nn*, you could use a mode 13 DBGET to skip backward in the list. A mode 12 DBGET returns condition 15 (end of chain). You can use either mode 12 or 13 to read forward or backward from the middle of a chain, as long as you do not reach the end or beginning of the chain.

Modes 15 and 16 read forward or backward through a chain of qualified records, without regard to chain boundaries. If you need to read past the beginning or end of a chain, modes 15 and 16 let you do so without returning a condition 14 or 15 (beginning of chain or end of chain). If the beginning or end of the file is reached, a condition of 10 or 11 is returned to the *status* array.

Mode values for reading chains of qualified records

Mode 25 and 26 read forward and backward through a chain of records qualified by DBFIND mode 12, 13, or 23. Records are returned in the order of their internal OMNIDEX ID. Mode 25 reads forward through the chain, and returns an end of chain condition after the last record is returned. Mode 26 reads backward through the chain, and returns a beginning of chain condition after the first record is returned.

Modes 22 and 23 let you skip forward or backward in a chain of qualified records. The number of records to skip is passed as a 32-bit integer in the *argument* parameter.

Mode 21 resets the pointer to the beginning of the list of qualified records.

If you use DBFIND mode 1 to find records, use DBGET mode 5 or 6 to read through the list of qualified records, just as you would for TurboIMAGE or sorted keys.

The internal ID list

After a successful call to DBFIND to search a keyword key, a list of one or more OMNIDEX IDs is accumulated. This internal ID list represents qualified records or record complexes. The OMNIDEX IDs in the internal ID list can take one of three forms, as discussed below.

Search items

After a DBFIND keyword search on a keyword key in a master set, or on a keyword key installed with the Record Complex (RC) option, the internal ID list consists of TurboIMAGE search items (SIs). After a DBFIND on a Record Complex key in a detail set, the qualifying count reflects the number of individual detail records that qualified, even though Record Complex keys qualify entire chains.

If the DBFIND keyword search was performed on a key in an unlinked (Detail Record indexed) detail set, the internal ID list consists of TurboIMAGE relative record numbers.

If the DBFIND keyword search was performed on a record specific (non-Record Complex) key in a linked detail set, the internal ID list consists of search item/relative record number combinations that represent the individual detail records of detail chains. You can use DBGET to retrieve the individual detail records.

Calling errors and exceptional conditions

This section lists the calling errors and exceptional conditions for both IMSAM and TurboIMAGE.

TurboIMAGE exceptional conditions

10	Beginning of file
11	End of file
12	Directed beginning of file
13	Directed end of file
14	Beginning of chain
15	End of chain
17	No entry found matching argument
18	Broken chain

OMNIDEX calling errors

-3300	Illegal mode specified
-3301	Data set not an IMSAM data set
-3304	Item not an IMSAM key

OMNIDEX exceptional conditions

3301	Critical flag set
3313	IMSAM tree empty

DBINFO

DBINFO (*base, qualifier, mode, status, buffer*)

DBINFO provides information about a database. This includes:

- what keys are installed on a data set
- the data type of a particular item
- the type of access permitted for each item
- information about Intrinsic Level Recovery

Parameters

base is the name of the integer array used as the base parameter when opening the database. The first element of the array must contain the base ID returned by DBIOPEN. (See "DBOPEN" in the *TurboIMAGE/XL Database Management System Reference Manual* for more information about the base ID.)

qualifier is the same as for current TurboIMAGE.

mode is a 16-bit word integer that determines the information that is returned.

There are eleven new modes that apply to OMNIDEX. The tables on the following pages list each of these modes, its purpose, and the contents of the *qualifier* and *buffer* parameters.

mode value	Purpose	*qualifier* contents	*buffer* array contents element		Comments
801	Returns information about the type of indexing product installed on a database	(ignored)	1 - 20	indexing product name (blank if none is installed)	
			21 - 25	version number of product	
			26	date of installation	
			27	not used	HP calendar date
			28	time of installation	
			29		Clock intrinsic format

Table 2-11: DBINFO mode values

mode value	Purpose	*qualifier* contents	*buffer* element	array contents	Comments
802	Returns number of internal and external index files created for database	(ignored)	1	number of external files	may be zero (0)
			2	number of internal files	may be zero (0)
			3	"G_" or "B_"	license for IMSAM license for OMNIDEX
			4-10	not used	
803	Indicates if third party indexing is enabled for a database	(ignored)	1	zero (0) or 1	zero (0) if indexing is disabled. 1 if indexing is enabled.
811	Describes access permitted to TurboIMAGE item or OMNIDEX key	word 1: data item (or OMNIDEX key) name or number followed by, word 9: data set name or number	1 - 8	OMNIDEX key name or number	OMNIDEX composite key numbers range from 1 to 9999 if TPI is not enabled. 10000 added to the defined item number indicates that TPI is enabled.
812	Describes OMNIDEX key's data type and structure (for compound items)	word 1: data item (or OMNIDEX key) name or number followed by word 9: data set name or number	1 - 8	OMNIDEX key name	This mode only works when TPI is enabled for the database
			9		
			10	data type (blank for composite key)	Valid data types are I, J, K, E, R, U, X, Z, and P.
			11	sub-item length	
			12	sub-item count	
			13	key type	0 if not an OMNIDEX key 1 if an OMNIDEX key only 2 if both a TurboIMAGE key and an OMNIDEX key

Table 2-11: DBINFO mode values

mode value	Purpose	*qualifier* contents	*buffer* array contents element		Comments
813	Identifies all items available in database, and type of access supported	(ignored)	(same as mode 103)		(same as mode 103)
814	Identifies all items available in referenced data set, and type of access supported	data set name or number	(same as mode 104)		(same as mode 104)
821	Identifies all data sets that contain the referenced OMNIDEX key	OMNIDEX key item name or number	1 2 . . *n*	a count of sets that contain that item ±set numbers of data sets that contain the referenced key	 This element repeats for as many sets as are returned in element 1. A negative set number indicates write access.
831	Identifies sorted (G + B) keys for specified data set	OMNIDEX data set name (UPPER CASE) or number	1 2 . . *n*	number of sorted keys (*n*) item number of sorted key	Applies as well to keys that are installed with both keyword and sorted access. This element consists of one 16-bit word and repeats for as many sorted keys are returned in element 1. 10000 added to the defined item number indicates that TPI is enabled.
832	Identifies keyword keys (M + B) for specified data set	OMNIDEX data set name (UPPER CASE) or number	1 2 . . *n*	number of keyword keys (*n*) item number of keyword key	Applies as well to keys that are installed with both keyword and sorted access. 10000 added to the defined item number indicates that TPI is enabled. Element 2 consists of one 16-bit word and repeats for as many keyword keys are returned in element 1.

Table 2-11: DBINFO mode values

mode value	Purpose	*qualifier* contents	*buffer* array element	contents	Comments
833	Provides information about referenced key	word 1: data set name or number, word 9: data item (or OMNIDEX key) name or number	1	item number of key (third party key number)	A negative value means that the key can be updated. Numbers greater than 10,000 represent composite keys.
			2	type of key (terminated with a space)	G = generic sorted M= keyword B= both types of access installed
			3	key length (bytes)	Contains zero (0) if referenced key is ungrouped
			4	set number of first item in key's group	Contains zero (0) if referenced key is ungrouped
			5	item number of first item in key's group	
			6	SI number of key's OMNIDEX SI	Elements 6 and 7 contain zero if search on key returns relative record numbers
			7	OMNIDEX SI length (bytes)	
			8	No Translate?	1 = yes, 0=no
			9	Parsing enabled?	1 = yes, 0=no
			10	any excluded words?	0 = no, 1 = excluded globally through an excluded words list.
			11	any blanks, nulls, or zeros indexed?	1 = yes, 0=no
			12	Native Language Translation? Batch Indexing?	0= uses binary collating 1= uses NLS collating
			13	Soundex?	1 = yes, 0=no
			14	Record Complex?	1 = yes, 0=no
			15		1 = yes, 0=no

Table 2-11: DBINFO mode values

mode value	Purpose	*qualifier* contents	*buffer* element	array contents	Comments
833	Provides information about referenced key	word 1: data set name or number, word 9: data item (or OMNIDEX key) name or number	16-21	reserved	
			22		
			23-26	OMNIDEX extensions	
			27	number of components in key	
			28	component item number	
			29	byte offset of item	elements 29-34 repeat for as many components as were returned in element 28.
			30	component length (bytes)	
			31		
			32	component data type	If the component was typecast during installation, that type is reflected.
			33	component sub-item length	
			34	component sub-item count	
834	Describes other keys grouped with the referenced key	word 1: data set name or number, word 9: (optional) followed by a data item (or OMNIDEX key) name or number	1	number of keys in group	
			2	set number of key in group	Elements 2 and 3 repeat for as many grouped keys as appear in element 1.
			3	item number of key in group	

Table 2-11: DBINFO mode values

buffer is the name of the array that contains the returned information

status is the name of an array of 10 16-bit words used to return information about the success of a call. See Table 2-1, on page 2-7, for a list of error condition codes.

Word 2 contains the number of words of information returned to the ***buffer*** parameter.

Calling errors and exceptional conditions

OMNIDEX calling errors

-3501 Dset not an IMSAM data set

-3502 Item not an IMSAM key

-3505 Not an OMNIDEX keyword field

Chapter 3: Programming

This chapter describes how to call the OMNIDEX and TPI interfaces from within application programs.

- "Writing Programs", on page 3-2, discusses some general aspects of programming, including opening the database, updating records and retrieving data by calling the OMNIDEX intrinsics. It also provides examples of 3GL applications through COBOL and FORTRAN.
- "Linking and Testing Programs", on page 3-33, discusses preparing programs, linking and testing programs, and program capabilities.
- "Interfacing with 3GLs", on page 3-40, presents COBOL source code examples of programs that perform various OMNIDEX retrievals. It is divided into two sections: one that presents OMNIDEX Intrinsic Interface examples, and one that presents Standard Interface to Third Party Indexing examples.
- "Interfacing with 4GLs", on page 3-60, discusses the 4GLs that interface with OMNIDEX.

Other publications, which discuss how to write OMNIDEX applications in a variety of programming languages, are available from DISC. These can be obtained for free by contacting your DISC sales representative or the DISC Response Center.

Before writing programs to access or update OMNIDEX-enhanced databases, programmers should be familiar with OMNIDEX's access capabilities. The "Intrinsics" chapter describes the syntax of the OMNIDEX and TurboIMAGE intrinsics, as called through your application programs.

Writing Programs

Introduction

OMNIDEX offers programmers a choice of developing applications using either of two sets of intrinsics:

- the OMNIDEX Intrinsic Interface
- the Standard Interface to Third Party Indexing, also known as "the TPI Interface"

This section discusses some basic functions of each of the two intrinsic interfaces mentioned above. It tells how to:

- open a database
- update records and the corresponding indexes
- retrieve records using OMNIDEX capabilities
- retrieve records using the Standard Interface to Third Party Indexing

The OMNIDEX Intrinsic Interface

The intrinsics for the OMNIDEX Intrinsic Interface reside in XLOMNIDX.PUB.SYS. Programs that use the OMNIDEX Intrinsic Interface exclusively should reference this library. See Appendix C of the *OMNIDEX ImagePlus SDK Administrator's Guide* for complete information about referencing XLs.

Opening databases

When developing OMNIDEX Intrinsic Interface applications, use the DBIOPEN intrinsic to open databases. This can be done by calling DBIOPEN directly in XLOMNIDX.PUB.SYS, or by calling either DBOPEN or DBIOPEN and referencing the Call Conversion library (XL.PUB.DISC) when linking a program, or in a RUN statement.

DBIOPEN performs two functions that are crucial for subsequent access to the database:

1. DBIOPEN calls DBOPEN to open the database.
2. DBIOPEN allocates space for the OMNIDEX local control blocks (OLCB and ILCB).

Updating data

When you update OMNIDEX-enhanced tables, you must also update any OMNIDEX indexes created for keys in those tables. Three OMNIDEX general intrinsics are used to update tables, and to automatically update any related indexes. They are:

DBIPUT	to add records
DBIUPDATE	to change data in existing records
DBIDELETE	to remove records

These OMNIDEX intrinsics correspond directly to the TurboIMAGE update intrinsics. The syntax for calling these intrinsics, which is discussed in the "Intrinsics" chapter, is identical to the syntax for calling TurboIMAGE update intrinsics. When programs are run through Call Conversion, any call to a TurboIMAGE update intrinsic is automatically converted into a call to its equivalent OMNIDEX general intrinsic.

The OMNIDEX update intrinsics have three mode options. Each of them provides a different method of updating keyword and sorted keys. These modes are:

- normal mode
- TurboIMAGE-only mode
- index-only mode

Normal mode

Normal mode is the most commonly used update mode. It is available for the DBIPUT, DBIUPDATE and DBIDELETE intrinsics.

By calling any of these intrinsics with mode 1, the specified data record and the OMNIDEX indexes are updated immediately. This is the recommended mode for online applications.

TurboIMAGE-only mode

TurboIMAGE-only mode is used when you want to update tables without updating their associated indexes. This mode is available for DBIPUT and DBIDELETE by adding 100 to the ***mode*** parameter value when calling these intrinsics.

This mode is typically used when performing large batch updates to reduce update overhead. After the data is added or updated in TurboIMAGE-only mode, the indexes must be reloaded using OmniUtil.

TurboIMAGE-only mode is not recommended for online applications unless keyword retrieval and sorted-sequential access on tables are not required immediately after updating. To selectively defer the updating of indexes for certain keys during normal mode updates, install the Batch Indexing (BI) option in them. See the *OMNIDEX ImagePlus SDK Administrator's Guide* for more information.

Index-only mode

Index-only mode is used when you want to update the OMNIDEX indexes without updating the actual record. This mode is available for DBIPUT and DBIDELETE only, and is specified by adding 200 to the ***mode*** parameter value when calling these intrinsics.

Error handling

Errors are handled differently for OMNIDEX applications than for TurboIMAGE applications. The most noticeable difference is OMNIDEX's use of a 21 word ***status*** array. This enables OMNIDEX intrinsics to handle OMNIDEX errors using word 11. TurboIMAGE errors still can be handled in the first 10 words of ***status***.

There are two types of error handling:

- A*ctive error handling*, the default, where errors incurred while updating OMNIDEX indexes are reflected in words 1 and 11 of the ***status*** parameter
- *Passive error handling*, where errors incurred while updating OMNIDEX indexes are reflected in words 11 and 12 of the ***status*** parameter, and remain transparent to a TurboIMAGE designed application

"Active error handling" is discussed first, and "passive error handling" is discussed last.

Active Error Handling

The words of the *status* parameter that are used to return OMNIDEX and TurboIMAGE error conditions are words 1-10, and 11. Word 1 is used as the TurboIMAGE condition word and the OMNIDEX error indicator. Words 1-10 are used to return information about TurboIMAGE errors. Word 11 is used as the OMNIDEX condition word. Programs commonly terminate when extraordinary errors occur, as indicated by the value in *status* Word 1. Most programs, especially those running through Call Conversion, treat OMNIDEX conditions 888 and 999 as extraordinary conditions.

The table below summarizes the use of the *status* parameter.

status word 1 value	*status* word 11 value	Indication
0	0	Successful execution; no errors and no warnings
+888	negative	OMNIDEX (keyword key) calling error
+888	positive	OMNIDEX (keyword key) exceptional condition
+999	negative	IMSAM (sorted key) calling error
+999	positive	IMSAM (sorted key) exceptional condition
other nonzero	0	Primary TurboIMAGE call error

Table 3-1: Status word values for normal (active) error handling

The various *status* words and their values under active error handling are discussed next.

Status word 1

Status word values correspond to the presence or absence of errors as follows:

If *status* word 1 (the TurboIMAGE condition word) is zero (0), the intrinsic call is successful and *status* word 11 (the OMNIDEX condition word) does not contain a valid error value.

If *status* word 1 is not zero (0), an error has occurred. Therefore, the TurboIMAGE condition word (word 1) should be checked for the success or failure of an OMNIDEX intrinsic.

There are three categories of errors: IMSAM (sorted key) errors, OMNIDEX (keyword key) errors and TurboIMAGE call errors.

- If an IMSAM error has occurred, *status* word 1 is set to +999.
- If an OMNIDEX error has occurred, *status* word 1 is set to +888. In either case, *status* word 11 is set to the value corresponding to the particular error.
- Otherwise, if *status* word 1 is nonzero, a TurboIMAGE error has occurred.

TurboIMAGE errors

Status word 1 contains the TurboIMAGE condition word. It is 0 (zero) if an OMNIDEX intrinsic executes successfully.

Status words 1-10 contain the *status* information returned in the primary or secondary TurboIMAGE call made by an OMNIDEX intrinsic.

OMNIDEX general intrinsics call TurboIMAGE intrinsics to update a TurboIMAGE entry or record. In a normal mode, for example DBIPUT calls DBPUT to add the entry passed in the *buffer* parameter. If this DBPUT is successful, various secondary calls to file system intrinsics are then made to update any associated indexes. If this DBPUT fails, a primary TurboIMAGE error is said to have occurred.

For example, referencing a nonexistent table (data set) in the *dset* parameter or passing a bad item list in the *list* parameter would yield a primary TurboIMAGE call error.

If one of the secondary intrinsic calls fails, a secondary error is said to have occurred. Some examples of secondary errors are when an index is full, the OMNIDEX root file is damaged, the database was structurally

altered without reinstalling OMNIDEX, or a problem exists in the IMSAM or OMNIDEX software.

In a secondary call failure, *status* words 1-10 contain the TurboIMAGE condition word resulting from the unsuccessful call, and *status* word 11 contains an OMNIDEX internal error code. The contents of *status* words 2-10 depend on the OMNIDEX intrinsic in question.

TPI Errors

Under Hewlett-Packard's Standard Interface to Third Party Indexing, it is possble to encounter OMNIDEX errors when using TurboIMAGE intrinsics to perform searches and updates to OMNIDEX databases. TPI errors are returned to the *status* array just like any other TurboIMAGE error.

TPI/OMNIDEX errors are distinguished from TurboIMAGE errors in that they use four-digit error that begin with the number "3". The remaining three digits correspond to an OMNIDEX error code. For example, if you were calling DBPUT to add a record to a data set and incurred error 3141, it is the same as a DBIPUT error `141 IMSAM tree is full`.

Status word 11: OMNIDEX and IMSAM Errors

If *status* word 1 is 888 or 999, and *status* word 11 (the OMNIDEX condition word) is also nonzero then an OMNIDEX or IMSAM error has occurred. The values for word 11 are discussed next.

Negative values for word 11 indicate *calling errors*. Calling errors are essentially syntactical, and result in the inability to execute a call successfully. For example, trying to read an entry in sorted-key sequence from an item that is not a sorted key would yield a calling error.

Positive values for word 11 indicate *exceptional conditions*. Exceptional conditions are incurred after a call has been executed. They prevent the return of the data requested in the call. Trying to read in sorted key sequence past the end of file would yield an exceptional condition error.

Each OMNIDEX intrinsic uses a specific range of condition word values to return exceptional conditions and calling errors in *status* word 11. Exceptional conditions use positive values and calling errors use negative values.

The values used by each intrinsic are:

Status **Word 11**	**Intrinsic Error**
100s	DBIPUT
200s	DBIFIND or ODXFIND
300s	DBIGET, ODXGET or ODXGETWORD
400s	DBIDELETE
500s	DBIINFO or ODXINFO
600s	DBIUPDATE
700s	Miscellaneous
800s	ODXTRANSFER or ODXVIEW
900s	DBIOPEN

A list of condition word values is included after the "Discussion" section of each intrinsic in chapter 2. Error condition word values and their messages are also listed in a file called ODXERROR in PUB.DISC. Call DBIERROR and DBIEXPLAIN to interpret OMNIDEX condition words.

Getting information about errors

To get help on any error condition, use OmniUtil's `Show Information` function. To run OmniUtil, enter the following run statement at a system prompt:

```
RUN OMNIUTIL.PUB.DISC
```

From the main menu, select `3. Show Information`. Then select `3. Messages`, and then, `1. OMNIDEX/IMSAM API`. Enter an error number or any word from an error message and OmniUtil displays a list of errors. You can get information about any error from the list by using your cursor keys to highlight it and pressing **[return]**.

If you are using OmniUtil to look up a 3000 series (TPI) error message, try entering the error number without the initial "3". For example, if you get error 3141 and there is no information listed for that number, enter 141 to get information about its OMNIDEX counterpart. The information listed for OMNIDEX errors is equally valid for their corresponding TPI errors.

If you receive a value that is not listed, it is an internal error and you should call the DISC Response Center at (303) 444-6610.

Passive Error Handling

Sometimes, third-party applications are used to update an OMNIDEX-enhanced database. These third-party applications are not equipped to interpret or handle OMNIDEX errors, especially indexing errors. So, when they are run through Call Conversion to automatically update the OMNIDEX indexes, and they incur an indexing error, they may terminate.

Passive error handling is *not* supported for TPI-enabled databases.

Passive error handling prevents OMNIDEX indexing errors from terminating a TurboIMAGE oriented application. This is accomplished by using *status* word 12 as the OMNIDEX error indicator. Because primary and secondary TurboIMAGE call errors are not returned to word 1, the calling application continues to run, even if an indexing error has occurred.

So, while OMNIDEX indexes may be corrupted, the application continues to update TurboIMAGE records. At the first indication of an error, an error message is sent to the console. Indexing errors are then logged to a passive error log file, as discussed next.

The use of the *status* parameter for passive error handling differs from active error handling in that word 12, not word 1, is used as the OMNIDEX error indicator. Table 3-2, on the opposite page, summarizes the use of the *status* parameter under passive error handling.

Word 12 value	Word 11 value	Indication
0	0	Successful execution; no errors and no warnings
+888	negative	OMNIDEX calling error
+888	positive	OMNIDEX exceptional condition
+999	negative	IMSAM calling error
+999	positive	IMSAM exceptional condition
other nonzero	0	Primary TurboIMAGE call error; data written to log file
other nonzero	positive	Secondary TurboIMAGE call error; data written to log file

Table 3-2: Status word values for passive error handling

Enabling or Disabling Passive Error Handling

To enable passive error handling you must set a flag in either the OMNIDEX Intrinsics XL (XLOMNIDX.PUB.DISC) or in the OMNIDEX call conversion XL (XL.PUB.DISC) depending on which XL your programs reference.

If you set the flag in XLOMNIDX, and copy XLOMNIDX to XLOMNIDX.PUB.SYS, you enable system-wide passive error handling for all programs that write to OMNIDEX indexed databases that are not TPI-enabled. By setting the flag in XL.PUB.DISC, only programs that reference the XL (or a copy of the XL) participate in passive error handling.

To enable (or disable) the passive error handling flag, perform the following steps:

1. Log on to the DISC account.
2. Run SETUP.PUB.
3. Choose menu option `6, Configure product library.`
4. Choose menu option `1, Configure Passive Error Handling.`
5. Enter the name of the library you want to configure.
6. Highlight the appropriate button to enable or disable PEH using the arrow keys, and press **[return]**.
7. Copy the XL to the appropriate location so that it is referenced by your programs.

Regardless of which XL is designated to contain the PEH flag, programs must reference XL.PUB.DISC for passive error handling to work.

The Error Log File

An error log file is a permanent file created for each database where passive error handling is installed. Its name consists of the database's name plus the suffix "LG". The file is created in the database's group. For example, the log file for a database named "SALES" that resides in the DEMODB group of the DISC account would be created as SALESLG.DEMODB.DISC.

The error log file is a 256 byte, 50,000 record, ASCII file. The record layout for the error log file is shown below.

Bytes	Length	Contents
1-8	8 bytes	session number
9-36	28 bytes	session name
37-66	30 bytes	program/process name
67-86	20 bytes	date stamp
87-96	10 bytes	time stamp
97-112	16 bytes	intrinsic name
113-140	28 bytes	database name
141-158	18 bytes	data set name
159-162	4 bytes	condition number (normally written to *status* word 1)
163-168	4 bytes	OMNIDEX/IMSAM error or exceptional condition number (normally written to *status* word 11)
169-240	72 bytes	OMNIDEX/TurboIMAGE error message

Table 3-3: The passive error log file

The error log file is automatically created the first time an error occurs. However, if a program is updating a database in an account other than the user's logon account, OMNIDEX cannot create the log file. This is because of an MPE restriction that prevents programs from building files across accounts. If programs log errors to a file in an account other than the program user's logon account, you must create the log file manually using the following BUILD statement:

```
BUILD logfile;REC=-256,1,F,ASCII;DISC=50000;CODE=7688
```

Logging Error Transactions

When passive error handling is enabled, and an internal OMNIDEX error occurs during a call to DBIOPEN, DBILOCK, DBIUPDATE, DBIPUT, DBIDELETE, or DBICLOSE, the first indexing error is echoed to the console and logged in the error log file, which is discussed later. The message looks something like this:

```
OMNIDEX communication from program: Error accessing
database dbname, data set setname. error message.
```

Subsequent errors are logged to the error log file only, and are not echoed to the console until the database is closed. The first error after the next DBIOPEN is again echoed to the console.

If an OMNIDEX index was corrupted as a result of an internal error, the passive error log file is created, flagging the database as having corrupt indexes.

Finding Indexing Errors

There are a variety of ways to tell if an index was corrupted under passive error handling. They are:

- Open a database. DBIOPEN always checks the error log file for a "corrupt" flag. If such a flag has been set, DBIOPEN returns a warning to the console:

  ```
  OMNIDEX communication from program: Previous
  indexing errors found for database dbname in
  file file.group.account.
  ```

- Check the error log file. The error log file provides a wealth of information about the date, time, affected data, and the nature of the errors that corrupted an index. The log file is discussed later.
- Call DBIINFO in mode 427. Mode 427 can be used to programmatically check to see if the indexes have been corrupted for any data set. See the DBIINFO listing of the Intrinsics chapter for more information about DBIINFO.

Correcting Indexing Errors

When a corrupt index is indicated by any of the three means described earlier, you must fix the index before performing any retrievals against its keys. To do this, use the OmniUtil Reindexing Options menu to reindex the entire database. This fixes any indexing discrepancies and purges the passive error log file.

OMNIDEX keyword retrieval

To do keyword retrievals on an OMNIDEX-keyed table (data set), use the OMNIDEX keyword retrieval intrinsics, ODXFIND and ODXGET.

1. Call ODXFIND to locate the keywords in the indexes and create a list of qualifying OMNIDEX IDs for the records that contain them.
2. Call ODXGET to retrieve the record identifiers for the qualifying records.
3. Call DBGET or DBIGET to retrieve each actual record using the OMNIDEX SI value or relative record number.

The following examples are common scenarios for databases enhanced with OMNIDEX. The required intrinsic calls are described for each. Note that the last example describes how to view documents in the document management system after using keyword retrieval to qualify them.

On a master set

Keyword retrieval on a master set depends on whether there is only one group or key, or several groups or keys.

One group or key

To retrieve records based on a keyword search against one group or key, perform the following intrinsic calls:

1. Call ODXFIND (mode 1, 2, 3, or 5) to find the records qualified by the keywords specified in the keyword list. The records are represented by their IDs, which are stored in memory as an internal ID list.
2. Call ODXGET (mode 1) to return the OMNIDEX SI values (TurboIMAGE search items) for the records that qualified to the calling application. The number specified in the *si-count* parameter tells ODXGET how many SIs to return. Repeat this call to return the next number (*si-count*) of OMNIDEX SIs. These SIs are passed to DBIGET or DBGET in the next step.
3. Call DBGET or DBIGET (mode 7) to return records to the calling application. The SIs returned from step 3 are passed through the *argument* parameter. Repeat this call for each OMNIDEX SI that is returned by ODXGET.

Several groups or keys

The program calls that are necessary for retrieval by multiple groups or fields are:

1. Call ODXFIND (mode 1, 2, 3, or 5)

   Returns the OMNIDEX IDs of records qualified by the keywords specified for the first group or key. These IDs are stored in memory as the *internal ID list*. ODXFIND also returns the number of records that qualify for each search to the *status* array.
2. Call ODXFIND (mode 1, 2, 3, or 5). Repeat for as many keys as you want to search.

   To continue a search across several fields, you must reload the internal ID list of the previous search. To do this, begin the keyword list with a leading AND or OR operator. A leading AND intersects the list of IDs qualified by the current ODXFIND (keyword search) with the IDs in the internal ID list. A leading OR unites the list of IDs qualified by the current ODXFIND (keyword search) with the IDs in the internal ID list. A leading AND NOT eliminates the list of IDs qualified by the current ODXFIND (keyword search) from the IDs in the internal ID list.

Note that all operations specified in the keyword list are performed before the leading Boolean operation. This qualifies records, which are then intersected with, united with, or eliminated from, the internal ID list, based on the leading Boolean operator.

After each search, ODXFIND returns a count of how many records qualified for the current search, and how many records are represented in the internal ID list.

A leading NOT operator begins a new search and overwrites the internal ID list.

3. Call ODXGET (mode 1) after you are finished searching for records.

   This returns the record identifiers (OMNIDEX SI values) of records that qualified. The number specified in the *si-count* parameter determines how many SIs are returned by each ODXGET. Repeat this call to return the next number (*si-count*) of OMNIDEX SIs. These SIs are passed to DBIGET or DBGET in the next step.

4. Call DBGET or DBIGET (mode 7) to return records to the calling application. The SIs returned from step 3 are passed through the *argument* parameter. Repeat this call for each OMNIDEX SI that is returned by ODXGET.

On a detail set

As with master sets, keyword retrieval on detail sets depends on whether you search one group or key, or several groups or keys. It also depends on whether the detail set contains record complex (RC) fields. Usually, keys in detail sets index keywords with each detail record's relative record number. Record complex keys, however, index keywords with the OMNIDEX master sets' SIs, as record complexes.

Default (record specific) keyword keys

Keyword retrieval on a detail set with normal keyword keys retrieves individual detail records and requires the following program calls:

1. Call ODXFIND (mode 1, 2, 3, or 5)

   Returns the IDs of detail records qualified by the specified keywords. The qualifying count returned to the *status* array contains the number of detail records qualified. You can call ODXFIND mode 30 to convert the IDs in the ID list (and the qualifying count) to reflect record complexes.

2. Call ODXGET (mode 11)

   Returns the relative record number for the first detail record represented in the ID list. Repeat this call for each detail record. The relative record numbers are then passed to DBGET or DBIGET in step 3.

3. Call DBGET or DBIGET (mode 4)

   Retrieves the first detail record. Repeat this call for each detail record.

To return search items to the calling application, either use ODXGET mode 1 or 21 (which could return duplicate SIs) or compress the ID list by calling ODXFIND mode 30 (which will not return duplicate SIs).

Unlinked (DR) detail set

Keyword retrieval on an unlinked detail set returns individual detail records and requires the following program calls:

1. Call ODXFIND (mode 1, 2, 3, or 5)

   Returns the IDs of detail records qualified by the specified keywords. The qualifying count returned to *status* reflects individual detail records. This count cannot be converted to record complexes.

2. Call ODXGET (mode 1)

   Returns the record number for the first detail record. Repeat this call for each detail record.

3. Call DBGET or DBIGET (mode 4)

   Retrieves the first detail record. Repeat this call for each detail record.

Record complex keys

Record complex keys on detail tables (data sets) index keywords and the record complexes that contain them. Therefore, record complex keys qualify the OMNIDEX master records (and their record complexes) of the details on which they are installed. The program calls necessary for keyword retrieval on record complex keys are:

1. Call ODXFIND (mode 1,2, 3, or 5)

   Returns the IDs of OMNIDEX record complexes (represented by their masters' SIs) qualified by the specified keywords. The qualifying count reflects the number of OMNIDEX masters (and their record complexes) that qualified.

2. Call ODXGET (mode 1)

   Returns the OMNIDEX SI values (the TurboIMAGE search items) for the record complexes that qualified.

   Because OMNIDEX returns the search item values, you must also read the chain defined by each search item to view individual detail record. The number in the *si-count* parameter determines how many SIs are returned by each ODXGET. Repeat this call to return the next number of OMNIDEX SIs as specified in *si-count.*

3. Call DBFIND or DBIFIND (mode 1)

   Uses the OMNIDEX SI value to set up a pointer at the chain head defined by the first SI. To perform a chained read of the detail records in that chain, go on to step 4. Repeat this call for each OMNIDEX SI value.

4. Call DBGET or DBIGET (mode 5)

   Retrieves each detail record in the chain. Repeat this call to read all entries in a chain. To read the next chain, repeat step 3.

Several record complex keys

The program calls for keyword retrieval on a detail using multiple groups or fields are:

1. Call ODXFIND (mode 1, 2, 3, or 5)

   Returns the number of OMNIDEX IDs (record complexes) qualified by the keywords specified for the first group or field.

2. Call ODXFIND (mode 1, 2, 3, or 5)

   To reload the internal ID list from the previous ODXFIND, begin the keyword list with a leading AND or OR operator, as described on page 3-14, followed by the keyword arguments you want to specify for the current key.

   Repeat this call for each additional group or key. This ODXFIND merges the internal ID list qualified by the previous ODXFIND with the IDs of records qualified in this call. Two qualifying counts are returned to the *status* array. One (words 7-8) reflects the number of records qualified by the keywords in this call, the other (words 3-4) reflects the number of records in the new internal ID list.

3. Call ODXGET (mode 1)

   Returns the OMNIDEX SI values (the TurboIMAGE search items) for the record complexes that qualified.

   Because OMNIDEX returns the search item values, you must also read the chain defined by each search item to view individual detail record. The number in the *si-count* parameter determines how many SIs are returned by each ODXGET. Repeat this call to return the next number of OMNIDEX SIs as specified in *si-count.*

4. Call DBFIND or DBIFIND (mode 1)

   Uses the OMNIDEX SI value to set up a pointer at the chain head defined by the first SI. To perform a chained read of the detail records in that chain, go on to step 5. Repeat this call for each OMNIDEX SI value.

5. Call DBGET or DBIGET (mode 5)

   Retrieves each detail record in the chain. Repeat this call to read all entries in a chain. To read the next chain, repeat step 4.

Other OMNIDEX retrievals

OMNIDEX also can retrieve only a qualifying count or retrieve only keywords.

Qualifying count

To retrieve only a qualifying count (not the records), the program call is:

Call ODXFIND (mode 1, 2, 3, or 5)

Returns the number of OMNIDEX IDs qualified by the keywords specified.

Qualify and retrieve only keywords

To retrieve only keywords, the program calls are:

1. Call ODXFIND (mode 10 or 11)

   Returns the number of keywords that qualify. *keywords* must contain a single range or generic keyword argument.

2. Call ODXGETWORD (mode 1 or 2)

   Returns the first keyword that qualified. Repeat this call to retrieve each qualifying keyword.

Multifind retrievals

Multifind lets you search across domains, and even databases, using data returned from one set of keyword searches (in the source set) as arguments against a key in the target set. This target set can reside in another domain or database.

Once the Multifind search is complete, an internal ID list is established in the target domain or database. Then, you can progressively qualify records by searching other keyword keys in the target domain.

Multifind from memory

To do Multifind retrievals from memory, an OMNIDEX key in the target domain must contain values common to the OMNIDEX search item of the source domain. For example, if the OMNIDEX SI of the source domain is PRODUCT-NO, you can retrieve records in another domain against a PRODUCT-NO keyword key.

To Multifind from memory:

1. Perform a keyword search as described above. To use Multifind from memory, you must qualify record complexes (SIs) as described for master tables (on page 3-13) and record complex keys (on page 3-17). You can also search the keyword keys of detail tables, but you must compress the ID list to reflect SIs, as described on page 3-16.
2. Call ODXFIND (mode 1, 2, 3, or 5) and reference a keyword key in the target domain.

   The *field* parameter references the keyword key in the target domain that corresponds to the OMNIDEX search item in the source domain. For example, PRODUCT-NO is the SI of the PRODUCTS master. It is also present as a keyword key in the ORDER-LINES detail, which is linked to the CUSTOMERS master (and domain). Therefore, you can use the PRODUCT-NO key of the ORDER-LINES detail as the target for a Multifind, after searching for PRODUCTS records.

   The *keywords* parameter should contain an ampersand (&) followed by a semicolon (;). This causes OMNIDEX to use the search items qualified in the first domain (PRODUCTS) as keyword arguments against the target key (PRODUCT-NO of ORDER-LINES).

   This call to ODXFIND establishes an internal ID list in the target domain (if the Multifind qualifies records).
3. You can progressively qualify records in the target domain by performing additional calls to ODXFIND on other keyword keys. Be sure to use a leading AND or OR, as described on page 3-14.

Multifind from a file

You can do Multifind retrievals using data from fields in the source set that are not OMNIDEX search items. To do this, you must first create a file containing data from a field in the source domain. This file can then be used as input for an OMNIDEX retrieval in the target domain.

1. Call ODXFIND (mode 1, 2, 3, or 5) on a key in the set of the source domain.

   Returns the number of OMNIDEX IDs qualified by the keywords specified. Repeat this call for all desired keyword groups or fields, using a leading AND or OR in the keyword list to reload the internal ID (record) list from the previous ODXFIND.

2. Call ODXTRANSFER (mode 1 or 2, or mode 101 or 102).

   To transfer values from a field that is also the OMNIDEX search item, call ODXTRANSFER mode 1 (for a new file) or 2 (to append to a file). Since OMNIDEX search items are stored internally, this process is extremely fast.

   To transfer values from a field other than the OMNIDEX search item, call ODXTRANSFER mode 101 (for a new file) or 102 (to append to a file). The *control* parameter of ODXTRANSFER should contain the table and item name of the field being transferred. This option is useful when the source of the data is an unlinked detail and the target is a master. Although OMNIDEX does not index the SIs of an unlinked detail, you can use ODXTRANSFER mode 101 to write them to a file for use against a keyword key installed on the SI of the corresponding master.

   When using ODXTRANSFER mode 101 or 102, the data transferred to the Multifind file can include anything that could establish a link between the source domain and the target domain. This could be dates, for example, or state abbreviations.

   Because OMNIDEX must retrieve each qualified master record to transfer the contests of the field referenced in *control*, mode 101 calls to ODXTRANSFER are much slower than mode 1 or 2 calls to ODXTRANSFER.

3. Call ODXFIND (mode 1, 2, 3, or 5) against a keyword key in the target domain that establishes a correspondence with the source domain.

   The *field* parameter should reference a key that contains values corresponding to the data transferred to the file. The *keywords* parameter should contain an ampersand (&), followed by the file name specified in the call to ODXTRANSFER, terminated by a semicolon (;).

   OMNIDEX uses the values contained in the file as keyword arguments against the target key.

4. You can progressively qualify records in the target domain by performing additional calls to ODXFIND on other keyword keys. Be sure to use a leading AND or OR, as described on page 3-14.

Sorted access

The following examples describe possible scenarios for partial-key and sorted-sequential retrievals against sorted keys, and the intrinsic calls for each. Note that although partial-key and sorted-sequential retrievals are described separately, they are usually combined.

In most sorted key retrievals, DBIGET is the only intrinsic used.

The only exception to this is using a sorted key that is also the search item in a master set. If you want to read the detail chain for a master record, you must first call DBIFIND to set up chain pointers to the detail(s).

For sorted key retrievals using DBIGET, it does not matter whether the sorted key is for a master set or a detail set.

The following examples show standard sorted key retrievals and sorted key retrievals that set up the chain head.

Note that mode 100 is =, 200 is >, 300 is >=, 400 is <, and 500 is <=.

Also note that for each partial-key retrieval using DBIGET or DBIFIND, the length of the partial-key value used in the search must be added to the mode. This is described after the examples.

Partial-key retrieval

To do a partial-key retrieval on a master set or detail set that contains a sorted key or composite key, the program calls are:

1. Call DBIGET (mode 100, 200, 300, 400 or 500 plus the number of words or bytes of the partial key value. A negative mode indicates bytes, a positive mode indicates words.)

   Returns a record to the specified buffer area of the calling program. This record is the first record in sequence that qualifies based on the partial-key value and the mode used.

Sorted sequential retrievals

To do sorted-sequential retrievals on a master or detail set that contains a sorted key or composite key, the program calls are:

1. Call DBIGET (mode 100, 200, 300, 400 or 500)

   Retrieves the first record in sequence and sets up a pointer to subsequent records.

2. Call DBIGET (mode 90 or 91)

   Retrieves the next record in sequence. Repeat this call to retrieve all the subsequent records in sequence by key.

Values are stored in a sorted key index as follows:

- For a sorted key that is also a TurboIMAGE search item in a master table (data set), the key stored in the index is the TurboIMAGE search item value.
- For a sorted key that is not a TurboIMAGE search item in a master table, the key stored in the index includes the sorted key value plus the TurboIMAGE search item value.

  For example, a master table might have an X2 field called STATE specified as a sorted key and an X4 TurboIMAGE search item called ACCT. If the values in these fields for a particular record are ACCT = 1234 and STATE = CO, the full internal key value in the index would be CO1234.
- For a sorted key in a detail table, the key stored in the index includes the sorted key value plus the relative record number.

  For example, if STATE was a sorted key and ACCT was the TurboIMAGE search item in a detail table, the key in the index would be CO plus the record number. If the record number was 978, the full key would be CO (978) where (978) is a double-word integer value.

  For more information about record numbers, see the DBGET intrinsic, mode 4, in the TurboIMAGE manual.

You must determine programmatically when to terminate the index sequential reads. DBIGET does not return an exceptional condition on a sequential read unless the beginning (i.e., the first key) or end (i.e., the last key) of a file is reached.

Setting up chain heads

To do a partial-key retrieval on a detail using a search item that is a sorted key in the associated master, the program calls are:

1. Call DBIFIND (mode 100, 200, 300, 400 or 500 plus the number of words or bytes of the partial value)

   Points to the first SI in sequence and calls DBFIND on that SI.

2. Call DBGET or DBIGET (mode 5)

   Retrieves the first detail record using a standard TurboIMAGE chained read. Repeat this call for all the records you want to retrieve from the chain.

The program calls to perform sorted-sequential retrievals on a detail set using a search item that is a sorted key in the associated master are as follows:

1. Call DBIFIND (mode 100, 200, 300, 400 or 500)

   Points to the first detail in a chain for the first qualifying key value.

2. Call DBIGET or DBGET (mode 5)

   Retrieves the first detail record using a standard TurboIMAGE chained read. Repeat this call for all the records in the chain that you want to retrieve.

3. Call DBIFIND (mode 90 or 91)

   Retrieves the next SI in sequence and calls DBFIND on that SI.

4. Call DBIGET or DBGET (mode 5)

   Retrieves the first detail record using a standard TurboIMAGE chained read. Repeat this call for all the records in the chain that you want to retrieve.

Using partial keys

When you do sorted partial-key retrievals, you must specify how many words or bytes are to be used for comparison during the sorted key search. To do this, add the length of the partial value specified by the user to the base mode value of 100, 200, 300, 400 or 500. A positive value specifies the length in words, while a negative value specifies the length in bytes. For example, 104 means the partial key is 4 words long, while a mode of -104 means the partial key is 4 bytes long.

For example, the following values might be indexed for a sorted key called DATE:

```
871215
880101
880202
890107
890108
890109
```

If you use the value 88 as an argument against the DATE sorted key, the following table illustrates which mode values would return which date values.

Mode value	Relational Operation	Key value retrieved
100 (denotes full key)	=	not found
101 (denotes generic)	=	880101
200 (denotes full key)	>	880101
201(denotes generic)	>	890107
300 (denotes full key)	>=	880101
301(denotes generic)	>=	880101
400 (denotes full key)	<	871215
401(denotes generic)	<	871215
500 (denotes full key)	<=	871215
501(denotes generic)	<=	880101

Table 3-4: Examples of DBGET relational modes

If you do a sorted key retrieval without specifying the word or byte length to compare for a partial value, an exact comparison is made on the full length of the key.

Index-only mode retrievals

Use index-only mode when you want to retrieve a sorted key without the corresponding record. To perform an index-only mode retrieval using DBIFIND or DBIGET, add 1000 to the mode value you would normally use. For example, to get the first key value greater than or equal to the partial key value "ABCD", the mode value would be 1302 (or -1304).

For example, you might have a TRANSACTIONS table in a general-ledger database with a sorted key called TRANS-KEY, which is a composite key comprised of the transaction DATE and AMOUNT. To find the transaction amounts for a particular date or month, you would not need to retrieve the data records themselves. All you would need are the key values.

For this retrieval, you could use 8701 as a partial value for the argument in an index-only mode DBIGET on the TRANS-KEY composite key.

This would return all the transactions for January 1987 in sorted-sequential order.

The calls would be:

1. Call DBIGET (mode 1102)

   Retrieves the first key in sequence and sets up a pointer to subsequent keys.

2. Call DBIGET (mode 1090)

   Retrieves the next key in sequence. This can be repeated for retrievals of the subsequent keys in sequence.

   The TRANS-KEY values retrieved for this composite key might be:

```
870101 100.01
870102 1874.56
870104 533.94
870110 12.50
870115 19820.34
```

When you only need key values, index-only mode retrievals improve performance tremendously. This is because sorted key indexes contain many keys in each physical record. For the above example, 3 I/Os would be required to retrieve the first 25 keys and 1 I/O would be required for every 25 keys thereafter.

Conversely, if you use normal mode instead of index-only mode for the above retrieval, it would require 3 I/Os for the first key and 1 I/O for each key thereafter. This amounts to 103 I/Os for 100 transactions using normal mode, versus only 6 I/Os for index-only mode. As you can see, index-only mode is much more efficient.

Standard Interface to Third Party Indexing

This section discusses how to perform certain functions, including OMNIDEX searches and retrievals, using the Standard Interface to Third Party Indexing, also known as "the TPI Interface". Programs that use the extended TurboIMAGE modes for Third Party Indexing (TPI) should resolve all TurboIMAGE calls through XL.PUB.DISC, or should access databases enabled for Third Party Indexing. Note that you must be using TurboIMAGE version C.04.03 or later to use the Standard Interface to Third Party Indexing on TPI-enabled databases.

Opening databases

When you open a database using DBOPEN, TurboIMAGE checks to see if the database is enabled for Third Party Indexing. If OMNIDEX indexes are enabled, DBOPEN calls the related OMNIDEX routine which will prepare other intrinsics to use the indexes.

Once a database is enabled for TPI, OMNIDEX indexing is automatically activated from TurboIMAGE intrinsics. To disable OMNIDEX parsing of DBFIND arguments for a specific application, add 100 to the DBOPEN mode.

If a database is not enabled for TPI, run any OMNIDEX application that access it through the Call Conversion library. The routines in this library route TurboIMAGE intrinsic calls to the OMNIDEX Intrinsic library, XLOMNIDX.PUB.SYS.

Updating data

Once the OMNIDEX product is installed and a database has been enabled for OMNIDEX indexing, OMNIDEX indexes are appropriately updated whenever a TurboIMAGE intrinsic is called. No other changes are needed by the system manager, the programmer or the user.

You can enable (or disable) OMNIDEX indexes for TPI (and automatic real-time indexing) using OmniUtil. This can be applied to individual databases, as discussed in "Enabling/disabling databases for Third Party Indexing (TPI)" in the "Topics" chapter of the *OMNIDEX ImagePlus SDK Administrator's Guide.*

Enabling real-time updates to indexes

You can enable the real-time updating of a database whose indexes are disabled for TPI. To do this, reference the Call Conversion XL (XL.PUB.DISC) in your update program's RUN statement, or link the program to XL.PUB.DISC using Hewlett-Packard's Link Editor. Programs that reference XL.PUB.DISC update indexes in real time, regardless of whether or not a database's indexes have been enabled for TPI. This also applies to databases that cannot be enabled for TPI because their version of TurboIMAGE does not support Third Party Indexing.

Error handling

The TurboIMAGE *status* array is still ten 16-bit words long. Errors are still handled by the DBERROR and DBEXPLAIN intrinsics.

The only difference you should be aware of is that there are new error codes and messages that relate to the new features enabled by OMNIDEX. These four digit error codes begin with the number "3," to distinguish them from standard TurboIMAGE error codes. The remaining three digits are identical to the digits that represent the same error in the OMNIDEX Intrinsic Interface. For example, if you were calling DBPUT to add a record to a data set and incurred error 3141, it is the same as a DBIPUT error `141 IMSAM tree is full`.

Third Party Indexing error codes are signed to indicate either a calling error, or an exceptional condition. As with standard TurboIMAGE codes, a negative value indicates a calling error, a positive value indicates an exceptional condition. These OMNIDEX calling error numbers are listed for each TurboIMAGE intrinsic in the "Intrinsics" chapter.

The TPI related condition values in TurboIMAGE are summarized below by the TurboIMAGE intrinsics that failed:

Status word 1	Intrinsic error
3100	DBPUT
3200	DBFIND
3300	DBGET
3400	DBDELETE
3600	DBUPDATE
3900	DBOPEN

Please note that passive error handling is not supported for TPI-enabled databases. For more help with errors, see "Error handling", on page 3-4.

OMNIDEX keyword retrieval

The TPI Interface does OMNIDEX keyword retrieval much like the OMNIDEX Intrinsic Interface. The DBFIND intrinsic is used in the same way as the ODXFIND intrinsic, and the DBGET intrinsic is similar to the ODXGET intrinsic. The major differences are:

- DBFIND keyword retrievals use a different order of precedence for Boolean operations than ODXFIND modes 1, 2 or 3. The DBFIND precedence is NOT, AND, OR.
- DBGET returns qualifying records, while ODXGET returns only search items. If DBGET is used on a detail table after a DBFIND on a record complex key, it locates the detail chain heads defined by the qualifying search items. After an ODXGET call, the application program determines whether to use the search item to retrieve the master record or the detail records in the chain.
- When performing keyword retrievals in DBFIND, terminate the *argument* parameter with a semicolon (;).
- The pound sign (#) is parsed as a wildcard. If the argument includes a pound-sign, enclose it in double quotes (").
- Spaces are parsed as AND operators. If the argument includes a space, enclose it in double quotes (").
- When searching integer keyword fields, the argument value should be in character format.

To do a keyword retrieval against a keyword key using the TPI Interface, call DBFIND and DBGET using the new retrieval modes discussed on page 2-101 and on page 2-117 respectively.

The sequence of intrinsic calls is:

1. Call DBFIND to locate keywords and qualify the records that contain them (based on the operations specified in the *argument* parameter). DBFIND stores a list of OMNIDEX IDs that represent the qualified records.
2. Call DBGET to retrieve the records identified by the internal ID list.

One group or field

Whether retrieving on a master or detail table (data set), the intrinsic calls and programming logic are essentially the same. The intrinsic calls to search a single OMNIDEX key are:

1. Call DBFIND (mode 12 or 23)

   This qualifies records based on the keywords and operations specified in DBFIND's *argument* parameter. Mode 12 also returns a qualifying count, while mode 23 does not. Since there is no performance penalty in returning the qualifying count, mode 12 is recommended.

   You could also use DBFIND mode 1, but for dedicated OMNIDEX applications, mode 12 or 23 is recommended.

2. Call DBGET (mode 25 or 26)

   Retrieves a record from the internal list of all qualified records. Mode 25 reads forward through the list, while mode 26 reads backward.

If the program reads to the beginning or the end of the qualified list of records, a "beginning of chain" or "end of chain" condition is returned. Consequently, programming for record retrieval on a single OMNIDEX key is done the same as a TurboIMAGE chained read.

Multiple groups or fields

To program for keyword retrievals on more than one key, make additional calls to DBFIND for each key or group to be searched.

1. Call DBFIND (mode 12 or 23)

   Qualifies records based on the keywords specified in DBFIND's *argument* parameter. Mode 12 returns a qualifying count, while mode 23 does not. Since there is no performance penalty for returning the qualifying count, mode 12 is recommended.

2. Call DBFIND (mode 12 or 23)

   Further refines the current (internal ID) list of qualified records based on the keywords and operations specified in DBFIND's *argument* parameter. The *argument* parameter should begin with an AND or OR operator. This indicate that the IDs found in this search should be either intersected (AND) or united (OR) with the IDs of the previous search, as contained in the internal ID list.

3. Call DBGET (mode 25 or 26)

   Retrieves a record from the internal ID list of qualified records. Mode 25 reads forward through the list, while mode 26 reads backward.

Sorted access

Retrieving records using sorted keys (IMSAM) is different under the TPI Interface than the OMNIDEX Intrinsic Interface. The main differences are:

- The argument passed in the DBFIND intrinsic defines the boundaries of the entire retrieval. In the OMNIDEX Intrinsic Interface, a call to DBIGET using an argument value merely establishes the starting point for a retrieval. It is left to the application program to decide where to terminate the retrieval.
- The DBFIND intrinsic can return a count of the number of records which meet the search criteria.
- Generic or partial key retrievals are specified by using the at sign (@) character, as opposed to the OMNIDEX Intrinsic Interface which uses the DBIGET mode value to specify a partial key retrieval.
- Sorted key retrievals using DBFIND support pattern matched retrievals using these "wildcard" characters: #, ?, @.

To retrieve records on IMSAM sorted sequential keys, the programming constructs are similar to the single OMNIDEX key retrieval:

1. Call DBFIND modes 1[1], 11, 21, or 22 to locate either the first key or all keys that qualify for the retrieval criteria. Modes 11 and 22 are used for binary keys (for example, TurboIMAGE type I, J, K, P, R, and E). Modes 1 and 11 return a qualifying count while modes 21 and 22 do not. There is a performance penalty for returning the qualifying count, so if the count is not needed, modes 21 or 22 are recommended.
2. Call DBGET modes 5, 6, 15, or 16 to retrieve the records that correspond to the qualifying keys. Modes 5 and 15 read forward in the sorted sequential order, while modes 6 and 16 read backward. Modes 5 and 6 return an "end of chain" or "beginning of chain" condition when the last or first qualified key in the chain is reached. Modes 15 and 16 read in sorted sequential order, regardless of the chain boundaries or pattern matching characters used. Modes 15 and 16 return an "end of file" or "beginning of file" condition when either of these boundaries is reached.

1. To use DBFIND mode 1 to perform a sorted search on a search item in a detail data set, you must first set the IMAGETPI job control word to 400.

Linking and Testing Programs

Introduction

There are some considerations when developing applications that use OMNIDEX or TurboIMAGE (TPI) intrinsics, or for adapting any program to access an OMNIDEX-enhanced database.

- You should consider whether to use the OMNIDEX Intrinsic Interface or the Standard Interface to Third Party Indexing. See the "Intrinsics" chapter for detailed information about either interface.
- Second, you should decide which libraries your programs should reference. This is discussed next with regard to the type of intrinsic interface you've selected, OMNIDEX or TurboIMAGE, and whether your programs run in compatibility mode.
- Finally, you may need to assign Multi-RIN (MR) capability to the program, and to the group and account where the program resides.

Linking and testing native mode programs

After you have written or modified your program to perform OMNIDEX retrievals, you should test it to ensure that your routines perform properly. To do this, you must reference the appropriate libraries to make the OMNIDEX routines available to your program.

The libraries you need to reference depend on the intrinsics you call, and whether the database is enabled for TPI. The XL setup for each of the scenarios discussed below is detailed in Appendix C of the *OMNIDEX ImagePlus SDK Administrator's Guide*. However, the simplest way to test your programs no matter what the situation is to reference the OMNIDEX Call Conversion library XL.PUB.DISC.

You can call the "DBI" and "ODX" intrinsics in the OMNIDEX Intrinsic Interface, or you can call TurboIMAGE intrinsics using the extended modes of the Standard Interface to Third Party Indexing. If you are on TurboIMAGE C.04.03 or later, or MPE/iX version 4.0 or later, you can use the Standard Interface to Third Party Indexing.

Which library your applications should resolve through depends on whether they access databases that are enabled for Third Party Indexing, or databases that are not enabled for TPI. Databases on earlier versions of TurboIMAGE are considered to be disabled for TPI. The various procedure libraries for OMNIDEX and TurboIMAGE are discussed below.

The OMNIDEX Call Conversion Library

The OMNIDEX Call Conversion library (XL.PUB.DISC) traps calls to TurboIMAGE or OMNIDEX intrinsics, and directs them to the appropriate library, depending on whether or not the database is enabled for TPI. The following scenarios discuss how the Call Conversion library traps and resolves calls:

OMNIDEX Intrinsic Interface, disabled database

Calls to "DBI" and "ODX" intrinsics in the OMNIDEX Intrinsic Interface are directed to XLOMNIDX.PUB.SYS, the main OMNIDEX Intrinsics library.

OMNIDEX Intrinsic Interface, enabled database

All activities are directed to XLOMNIDX.PUB.SYS. XLOMNIDX directs other TurboIMAGE activities, such as opening and locking the database or updating the database, to the TurboIMAGE intrinsics in XL.PUB.SYS. XL.PUB.SYS handles indexing through the Standard Interface to TPI, which calls routines in XLOMNIDX.PUB.SYS.

Standard Interface to TPI, enabled database

When a database is enabled for TPI, you can call TurboIMAGE intrinsics with the extended TPI modes, and no special library references are required. OMNIDEX indexing and retrieval are automatically invoked by TurboIMAGE through the Standard Interface to TPI. If you reference the OMNIDEX Call Conversion library, however, the TurboIMAGE calls are passed from XLOMNIDX to the TurboIMAGE intrinsics in XL.PUB.SYS.

Note that if a program accesses a database that is enabled for TPI, it is not necessary to run it through Call Conversion. Such programs can resolve through the system XL.

Standard Interface to TPI, disabled database

You can use TurboIMAGE's extended TPI modes against databases that are not enabled for Third Party Indexing. Such calls to TurboIMAGE retrieval intrinsics are trapped by the Call Conversion library and converted to call the appropriate OMNIDEX intrinsics, like ODXFIND, ODXGET, DBIFIND or DBIGET. Similarly, other TurboIMAGE general intrinsic calls are converted to their OMNIDEX general intrinsic counterpart. For example, calls to DBPUT are converted to call DBIPUT. This same technique was used to implement the TurboIMAGE Retrieval Interface, introduced in version 2.09/2.10 of OMNIDEX.

Referencing the Call Conversion library

The easiest way to reference the OMNIDEX Call Conversion library is with the MPE/iX XL path. This can be done when the program is linked, or when the program is run. Establishing the XL path at link time might look like this:

```
LINK $OLDPASS,myprog;XL="XL.group.account,XL.PUB.DISC"
```

Establishing the XL path at run time might look like this:

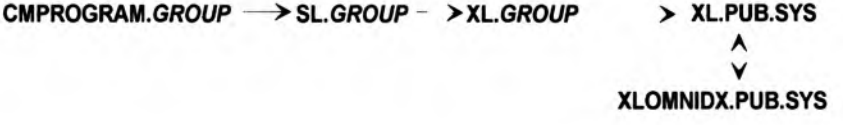
```
RUN myprog;XL="XL.group.account,XL.PUB.DISC"
```

Note that you should reference your own application XLs before the OMNIDEX Call Conversion XL.

Testing compatibility mode applications

As with native mode, any compatibility mode application that resolves TurboIMAGE or OMNIDEX procedure calls through the OMNIDEX Call Conversion library will always function properly. It does not matter whether the program uses the OMNIDEX Intrinsic Interface or the Standard Interface to Third Party Indexing. Neither does it matter whether or not the database accessed by the application is enabled for TPI.

For compatibility mode OMNIDEX and TurboIMAGE calls to pass through the OMNIDEX Call Conversion Library, compatibility mode programs must reference the OMNIDEX Intrinsic Switch Stub library (SL.PUB.DISC). The Intrinsic Switch Stub library traps compatibility mode OMNIDEX and TurboIMAGE calls, and directs them as native mode calls to the Call Conversion library. To facilitate this process, place the routines in the OMNIDEX Call Conversion library and the routines in the Intrinsic Switch Stub library in the same group as the calling application. Run the program with LIB=G to reference the Intrinsic Switch Stub library. If the application resides in a PUB group, you can run it with either ;LIB=P or ;LIB=G. The flow of calls is:



Adding Switch Stub routines to existing SLs

If an SL already exists in the group where you want to place the Intrinsic Switch Stub routines, use SEGMENTER to copy the Switch Stub segments from USL.PUB.DISC into the existing SL. To add the Switch Stub USL segments to an existing SL, perform the following steps:

1. Log on to the group that contains the SL.
2. Use Hewlett-Packard's SEGMENTER to add the OMNIDEX Intrinsic Switch Stub segments to the existing SL. For example:

```
:SEGMENTER

-SL SL
-USL USL.PUB.DISC
-ADDSL DISC'2
-ADDSL DISC'CM'SWITCH
-ADDSL DISC'4
-ADDSL DISC'7
-ADDSL DISC_C
-ADDSL CCSCSEG0
-ADDSL CCSCSEG1
-ADDSL CCSCSEG2
-ADDSL UTIL'1
-ADDSL IMAGE'CC
-EXIT
```

Adding Call Conversion routines to existing XLs

If there is already an XL in the group where you are installing the Switch Stub and Call Conversion procedures, you can merge them with the existing XL. To add the Call Conversion XL routines to an existing XL, perform the following steps:

1. Log on to the group that contains the XL.
2. Run Hewlett-Packard's Link Editor by typing the LINKEDIT command:

```
:LINKEDIT
```

3. Copy the Call Conversion XL into the existing XL using the COPYXL command:

```
LinkEd> COPYXL FROM=XL.PUB.DISC;TO=XL
```

4. Exit LINKEDIT

```
LinkEd> EXIT
```

Then, simply run the program with LIB=G

```
RUN myprog;LIB=G
```

Note that if a TurboIMAGE program accesses a database that is enabled for TPI, it is not necessary to reference any OMNIDEX libraries when linking or running the program. Compatibility mode program calls resolve through the system SL, like this:

Program capabilities

When databases are enabled for TPI, no special program capabilities are required. However, when updating databases that are not enabled for TPI, Multi-RIN (MR) capability is required if the program also performs TurboIMAGE locking. Multi-RIN capability permits OMNIDEX to lock index files at the same time that locks are held against TurboIMAGE data sets. MR capability is typically added at the time the program is linked.

Processes that use MR should lock at the same resource level, and in the same order. See Appendix D of the *TurboIMAGE/XL Database Management System Reference Manual* for more information about MR and avoiding deadlocks.

If you do not want to re-link programs with MR capability, or you are unable to because you do not have the source files, you can add MR capability to native mode or compatibility mode programs using FUTIL.UTIL.DISC. An example of using FUTIL to add MR capability to all programs in a group follows.

FUTIL is in the UTIL group of the DISC account. To run it, enter:

```
RUN FUTIL.UTIL.DISC
```

The prompts and sample responses for adding MR capability are shown below. Note that the FUTIL MODIFY command supports MPE wildcard characters (@, ?, #). This lets you add account capabilities to multiple program files, for example:

```
Operation (Copy/Modify/Purge/Rename/Exit) ? M
Enter file name (@,?,# wildcards ok)? @.PUB
Maxdata (for MPE V programs)? 31232
Capabilities ? IA, BA, PH, MR
```

Note that when modifying a single file, current values are displayed in brackets at each prompt. When adding program capabilities, you must re-specify the existing program capabilities to keep them. Pressing **[return]** at the `Capabilities ?` prompt leaves the program capabilities unchanged.

To exit FUTIL, enter E at the Operation prompt:

```
Operation (Copy/Modify/Purge/Rename/Exit) ? E
```

After you have added MR capability to one or more programs, be sure that the account and group where the programs reside also have MR capability.

Interfacing with 3GLs

The advanced retrieval capabilities of OMNIDEX and IMSAM can be called easily through third generation languages (3GLs) like BASIC, COBOL, FORTRAN, PASCAL, C, SPL and RPG.

The previous section of this chapter outlined the sequence of the intrinsic calls you would use within an application program. The Intrinsics chapter discusses the syntax for these intrinsic calls.

This section describes the parameters you use when you call the OMNIDEX intrinsics. It also provides examples of COBOL programs so you can see how these calls are used.

If you would like to see examples of OMNIDEX applications written in PASCAL, FORTRAN, BASIC or RPG, refer to the *OMNIDEX Language Sampler*. Source files of programs which call OMNIDEX intrinsics through BASIC, COBOL, FORTRAN and PASCAL are also available in the DEMO group of your DISC account.

COBOL program examples (OMNIDEX Intrinsic Interface)

This section provides examples of COBOL code for each OMNIDEX retrieval intrinsic.

Note that the following examples are only partial listings of COBOL programs. Use the sample programs in the DEMO group of the DISC account for more complete examples of interfacing COBOL with OMNIDEX.

Also note that the SPL notation for comments (<< comment >>), which is occasionally used in these examples, is not allowed in COBOL.

DBIFIND

Use DBIFIND only when a sorted key is also a TurboIMAGE search item in a master, and you require subsequent retrieval of detail records.

```
PROGRAM-ID. programname.
          .
          .
 01 RECORD-BUFFER.
          .
          .
 01 DSET.
          03 DATA-SET          PIC X(16).
          03 IMSAM-key         PIC X(16).
          .
          .
 01 ITEM                      PIC X(16).
          .
          .
 PROCEDURE DIVISION.
          .
          .
          .
100-FIND-RECORD.
          .
          .
          MOVE "IMSAMDETAILSET;" TO DATA-SET.
          MOVE "IMSAMKEY;" TO ITEM.
<< Note that the key must be a TurboIMAGE search
item >>
          MOVE -305 TO MODE.
<< Assume a 5-byte partial key >>
          MOVE "PARTIALKEY" TO ARGUMENT.
          CALL "DBIFIND" USING BASE, DSET, MODE,
                STATUS, ITEM, ARGUMENT.
          IF IMAGE-STATUS NOT EQUAL 0 THEN
                IF IMS-ODX-STAT =210
                OR IMS-ODX-STAT =211
                OR IMS-ODX-STAT =217 THEN
                DISPLAY "Key value not found"
          ELSE
                PERFORM ERROR-ROUTINE
                GO TO END-OF-PROGRAM.
```

```
200-GET-RECORD.

        MOVE 5 TO MODE.
        CALL "DBIGET" USING BASE, DSET, MODE,
              STATUS, LIST, BUFFER, ARGUMENT.
        IF IMAGE-STATUS = 15 THEN
              GO TO 300-FIND-NEXT.
        IF IMAGE-STATUS NOT EQUAL 0 THEN
              PERFORM ERROR-ROUTINE
              GO TO 100-FIND-RECORD.

        DISPLAY RECORD-BUFFER.
        GO TO 200-GET-RECORD.


300-FIND-NEXT.

        MOVE 90 TO MODE.
        "DBIFIND" USING BASE, DSET, MODE,
              STATUS, ITEM, ARGUMENT.
        IF IMAGE-STATUS NOT EQUAL 0 THEN
              IF IMS-ODX-STAT = 211 THEN
                  DISPLAY "End of file"
                  GO TO END-OF-PROGRAM
              ELSE
                  PERFORM ERROR-ROUTINE
              GO TO END-OF-PROGRAM.
        GO TO 200-GET-RECORD.
        .
        .
        .
ERROR-ROUTINE.
        CALL "DBIEXPLAIN" USING STATUS, PARM.
        << Note additional parameter! >>
```

DBIGET

Use DBIGET for all types of sorted keys in both master and detail tables. Refer to the file COBIMSS.DEMO.DISC for a sample program that shows normal mode and index-only mode calls to DBIGET.

```
PROGRAM-ID. programname.
        .
        .
 01 RECORD-BUFFER.
        .
        .
 01 DSET.
        03 DATA-SET          PIC X(16).
        03 IMSAM-KEY         PIC X(16).
        .
        .
 01 ITEM                    PIC X(16).
        .
        .
 PROCEDURE DIVISION.
        .
        .
100-GET-RECORD.
        .
        .
        MOVE "IMSAMDATASET;" TO DATA-SET.
        MOVE "IMSAMKEY;" TO IMSAM-KEY.
        MOVE -305 TO MODE.<<Assuming 5 byte partialkey>>
        MOVE "PARTIALKEY" TO ARGUMENT.
        CALL "DBIGET" USING BASE, DSET, MODE,
             STATUS, LIST, BUFFER, ARGUMENT.
        IF IMAGE-STATUS NOT EQUAL 0 THEN
              IF IMS-ODX-STAT = 310
              OR IMS-ODX-STAT = 311
              OR IMS-ODX-STAT = 317 THEN
                  DISPLAY "Key value not found"
                  GO TO 100-GET-RECORD
              ELSE
                  PERFORM ERROR-ROUTINE
                  GO TO END-OF-PROGRAM.
        .
        .
ERROR-ROUTINE.
        CALL "DBIEXPLAIN" USING STATUS, PARM.
```

ODXFIND

Refer to the file COBODXS.DEMO.DISC for another program example that shows OMNIDEX keyword retrievals using ODXFIND (and ODXGET).

```
PROGRAM-ID. programname.
          .
          .
100-BEGIN.
          .
          .
          MOVE 1 TO MODE.
          MOVE "DSETNAME" TO DSET.
          MOVE "FIELDNAME" TO ITEM.
          MOVE "KEYWORD1,KEYWORD2,..." TO KEYWORDS.
               << Terminate list with space or semicolon >>
          CALL "ODXFIND" USING BASE, DSET, MODE,
               STATUS, ITEM, KEYWORDS.
          IF IMAGE-STATUS NOT EQUAL 0 THEN
               IF IMS-ODX-STAT = 217 THEN
                   DISPLAY "Keyword not found"
                   GO TO 100-BEGIN
               ELSE
                   PERFORM ERROR-ROUTINE
          ELSE
               DISPLAY REC-COUNT.


200-GET-RECORDS.
          .
 <<Described on next page>>
          .
ERROR-ROUTINE.
          CALL "DBIEXPLAIN" USING STATUS, PARM.
```

ODXGET

ODXGET requires an internal ID list that contains record IDs qualified in a keyword search (an ODXFIND with a mode of 1, 2, 3, or 5). Refer to the file COBODXS.DEMO.DISC for an example of OMNIDEX keyword retrievals using ODXFIND (and ODXGET).

```
PROGRAM-ID. programname.
          .
          .
 01 SI-LIST.
          03 SI-LIST-ID       PIC S9(9) COMP.
          .
          .
200-GET-RECORDS.
          MOVE 1 TO SI-COUNT.
          MOVE 1 TO MODE.
          CALL "ODXGET" USING BASE, MODE, STATUS,
                SI-LIST, SI-COUNT.
          IF IMAGE-STATUS NOT EQUAL 0 THEN
          IF IMS-ODX-STAT = 311 THEN
                DISPLAY "End of qualifying records"
                GO TO 100-BEGIN
          ELSE
                PERFORM ERROR-ROUTINE
                   GO TO 100-BEGIN
          ELSE
                PERFORM 300-GET-RECORDS.
300-GET-RECORDS.

          MOVE SI-LIST-ID TO ARGUMENT.
          MOVE 7 TO MODE.
          MOVE "DATASETNAME" TO DSET.
          CALL "DBGET" USING BASE, DSET, MODE,
                STATUS, LIST, BUFFER, ARGUMENT.
<< Note that DBGET is used for the IMAGE mode 7 Get >>
          IF IMAGE-STATUS NOT EQUAL 0 THEN
          .
          .
          ELSE
                PERFORM 400-DISPLAY-REC.
          .
          .
 ERROR-ROUTINE.
          CALL "DBIEXPLAIN" USING STATUS, PARM.
```

ODXGETWORD

ODXGETWORD requires a preceding call to ODXFIND with a mode of 10 or 11.

```
PROGRAM-ID. programname.
          .
          .
200-GET-KEYWORDS.
          .
          .
          MOVE 2 TO MODE.
          CALL "ODXGETWORD" USING BASE, MODE, STATUS,
                TARGET.
          IF IMAGE-STATUS NOT EQUAL 0 THEN
                PERFORM ERROR-ROUTINE
                GO TO 200-GET-KEYWORDS.
          DISPLAY TARGET, STATUS-WORDS-12-13.
          GO TO 200-GET-KEYWORDS.
          .
          .
ERROR-ROUTINE.
          CALL "DBIEXPLAIN" USING STATUS, PARM.
```

ODXTRANSFER

Refer to the file COBODXS.DEMO.DISC for an example.

```
PROGRAM-ID. programname.
        .
        .
 01 FILENAME              PIC X(06) VALUE "ODXIDS ".
        .
        .
 TRANSFER-ENTRIES.
        CALL "ODXTRANSFER" USING DBN-PATH1, MODE1,
        DB-STATUS,
              FILENAME, ODXTR-OPT.
        IF IMAGE-STATUS NOT = 0 THEN
              PERFORM ERROR-DISPLAY
              MOVE YES TO DONE.
              GO TO TRANSFER-ENTRY-EXIT.

        PERFORM CREATE-REPORT THRU CREATE-REPORT-EXIT.
 TRANSFER-ENTRY-EXIT.
 EXIT.
```

The report that uses the search item values from the ODXTRANSFER is as follows:

```
CREATE-REPORT.
        OPEN INPUT ODXID.
        OPEN OUTPUT REPORT-TR.
        MOVE SPACES TO FILLER1.
        MOVE SPACES TO FILLER2.
        MOVE SPACES TO FILLER3.
        READ ODXID RECORD INTO OMNSIEX-SI AT END
        MOVE YES TO DONE.
        PERFORM PRINT-RECORD THRU PRINT-RECORD-EXIT
              UNTIL DONE = YES.
        CLOSE ODXID.
        CLOSE REPORT-TR.
 CREATE-REPORT-EXIT.
        EXIT.
```

The contents of a field can be transferred to a file using mode option 200, as illustrated in the example on the following page. Note that *dsetname* (in RETRIEVE-SAMELIST) must be the same set that was originally used in the first ODXFIND (in GET-LIST). The *fieldname* can be any valid keyword key in the set.

```
PROGRAM-ID. programname.
.
.
01 ODX-FILENAME.
  03 FILLER                    PIC X VALUE "$".
  03 FILENAME                  PIC X(06)VALUE "ODXIDS".
.
.
 GET-LIST.
 MOVE "DSETNAME" TO DSET.
 MOVE "FIELDNAME" TO ITEM.
 MOVE "KEYWORD1,KEYWORD2,..." TO KEYWORDS.
 CALL "ODXFIND" USING DBN-PATH1, DSET, MODE1,
        DB-STATUS, ITEM, KEYWORDS.
 IF IMAGE-STATUS NOT = 0 THEN
        IF IMS-ODX-STAT = 217 THEN
              DISPLAY "Keyword not found"
              GO TO GET-LIST
        ELSE
              PERFORM ERROR-ROUTINE
 ELSE
        DISPLAY REC-COUNT.
.
.
TRANSFER-ENTRIES.<< Freeze the list of qualified records >>
 CALL "ODXTRANSFER" USING DBN-PATH1, MODE201,
        DB-STATUS, FILENAME, ODXTR-OPT.
 IF IMAGE-STATUS NOT = 0 THEN
        PERFORM ERROR-DISPLAY
        MOVE YES TO DONE.
.
.
RETRIEVE-SAMELIST.
 MOVE "DSETNAME" TO DSET.
 MOVE "FIELDNAME" TO ITEM.
 CALL "ODXFIND" USING DBN-PATH1, DSET, MODE1,
        DB-STATUS, ITEM, ODX-FILENAME.
 IF IMAGE-STATUS NOT = 0 THEN
        PERFORM ERROR-ROUTINE
 ELSE
        DISPLAY REC-COUNT.
```

COBOL program examples (TPI Interface)

This section provides examples of COBOL code for a keyword search and a sorted search using the Standard Interface to Third Party Indexing.

Note that the following examples are only partial listings of COBOL programs. Use the sample programs in the DEMO group of the DISC account for more complete examples of interfacing COBOL with OMNIDEX.

A sorted retrieval

This program performs a sorted retrieval against the key installed on the DATE-ENTERED field of the CUSTOMER-NOTES data set. The search would qualify a chain of records based on the argument entered at the `Key Value?` prompt. The working storage section has been abridged to eliminate the usual TurboIMAGE parameter definitions. Refer to the file COBIMS3S.DEMO.DISC for a complete listing of the source code.

```
$CONTROL USLINIT, OPTFEATURES = CALLALIGNED16, CALLINTRINSIC
.
.
 ENVIRONMENT DIVISION.
 DATA DIVISION.
 FILE SECTION.
 WORKING-STORAGE SECTION.
.
.

 01 CUST-NOTES-REC.
    03 CUSTOMER-NO          PIC S9(09) COMP VALUE 0.
    03 DATE-ENTERED         PIC X(6) VALUE SPACES.
    03 TICKLER              PIC X(2) VALUE SPACES.
    03 INITIALS             PIC X(4) VALUE SPACES.
    03 ACTION               PIC X(60) VALUE SPACES.

 01 KEY-VALUE.
    03 KEY-VALUE-IN         PIC X(7) VALUE SPACES.
    03 TERMINATOR           PIC X VALUE ";".
```

```
01 MISCELLANEOUS-FIELDS.
   03 ANSWER                PIC X(02) VALUE SPACES.
   03 ITEM-LIST             PIC X(02) VALUE "@;".
   03 CUST-NO-DISP          PIC Z(08)9.
   03 SEPARATOR             PIC X(72).

PROCEDURE DIVISION.

MAIN SECTION.
MOVE ALL "-" TO SEPARATOR.

MOVE 1 TO DBM-MODE.

CALL "DBOPEN" USING DBN-PATH,
DBP-WRITE,
DBM-MODE,
DB-STATUS.

IF IMAGE-STATUS NOT = 0 THEN
PERFORM EXPLAIN-DISPLAY
PERFORM ERROR-DISPLAY
DISPLAY " "
STOP RUN.

PERFORM REQUEST-KEY THRU REQUEST-EXIT
UNTIL KEY-VALUE-IN = "EXIT" OR "exit" OR "E" OR "e".

PERFORM CLOSE-DB.
```

The paragraph REQUEST-KEY-VALUE continues to execute until the user types EXIT or exit in response to `Key Value?`.

```
REQUEST-KEY.

 MOVE SPACES TO KEY-VALUE-IN.
 DISPLAY " ".
 DISPLAY "Please enter a date value or 'E' to exit.".
 DISPLAY "You may enter wildcard characters (@,#,?).".
 DISPLAY " ".

 ACCEPT KEY-VALUE-IN.

 IF KEY-VALUE-IN = "E" OR "e" OR "EXIT" OR "exit"
 GO TO REQUEST-EXIT.

 MOVE 1 TO DBM-MODE.

 CALL "DBFIND" USING DBN-PATH,
 DBSETS,
 DBM-MODE,
 DB-STATUS,
 IMSAM-KEY,
 KEY-VALUE.

 IF IMAGE-STATUS NOT = 0 THEN
 PERFORM ERROR-DISPLAY
 GO TO REQUEST-EXIT.

 MOVE "N" TO ANSWER.
 PERFORM GET-REC THROUGH GET-REC-EXIT
 UNTIL ANSWER = "EXIT" OR "exit" OR "E" OR "e".

REQUEST-EXIT.
 EXIT.
```

This part of the code calls DBGET to read through the chain of records that the user qualified using DBFIND. The user can enter N to view the next record in the chain, P to view the previous record in the chain, or E to exit back to the search prompt.

```
GET-REC.

    *
    * Mode 5 is a forward chained read. Mode 6 is a
    * backward chained read.
    *
     IF ANSWER = "N" OR "n"
            MOVE 5 TO DBM-MODE
     ELSE
            IF ANSWER = "P" OR "p"
                  MOVE 6 TO DBM-MODE
            ELSE
                  DISPLAY " "
                  DISPLAY "INVALID REPLY!"
                  DISPLAY " ".

     CALL "DBGET" USING DBN-PATH,
                        DBSETS,
                        DBM-MODE,
                        DB-STATUS,
                        ITEM-LIST,
                        CUST-NOTES-REC,
                        DUMMY.

     IF IMAGE-STATUS NOT = 0 THEN
            PERFORM ERROR-DISPLAY
     ELSE
            PERFORM DISPLAY-REC.

     MOVE SPACES TO ANSWER.

     DISPLAY " ".
     DISPLAY "Please enter an action:".
     DISPLAY " ".
     DISPLAY " 'N' to see the next record".
     DISPLAY " 'P' to see the previous record".
     DISPLAY " 'E' to exit".
     DISPLAY " ".

     ACCEPT ANSWER.

GET-REC-EXIT.
     EXIT.
```

```
DISPLAY-REC.

 MOVE CUSTOMER-NO TO CUST-NO-DISP.

 DISPLAY " ".
 DISPLAY SEPARATOR.
 DISPLAY "CUSTOMER NO : ", CUST-NO-DISP.
 DISPLAY "DATE ENTERED: ", DATE-ENTERED.
 DISPLAY "TICKLER : ", TICKLER.
 DISPLAY "INITIALS : ", INITIALS.
 DISPLAY "ACTION : ", ACTION.
 DISPLAY SEPARATOR.

EXPLAIN-DISPLAY.

 DISPLAY " ".
 CALL "DBEXPLAIN" USING DB-STATUS.
 DISPLAY " ".
 DISPLAY "PRESS RETURN TO CONTINUE.".
 ACCEPT ANSWER.

ERROR-DISPLAY.

 DISPLAY " ".
 CALL "DBERROR" USING DB-STATUS, ERROR-MSG, MESSAGE-LEN.
 DISPLAY " ".
 DISPLAY ERROR-MSG.
 DISPLAY " ".
 DISPLAY "PRESS RETURN TO CONTINUE.".
 ACCEPT ANSWER.

CLOSE-DB.

 MOVE 1 TO DBM-MODE.
 CALL "DBCLOSE" USING DBN-PATH,
                      DBSETS,
                      DBM-MODE,
                      DB-STATUS.


 IF IMAGE-STATUS NOT = 0 THEN PERFORM ERROR-DISPLAY.
 STOP RUN.
```

A keyword search and retrieval

This program performs a keyword retrieval against the key installed on the DATE-ENTERED field of the CUSTOMER-NOTES data set. The search would qualify records based on the arguments entered at `Key Value?` prompts. The working storage section has been abridged to eliminate the usual TurboIMAGE parameter definitions. Refer to the file COBODX3S.DEMO.DISC for a complete listing of the source code.

```
    03 LAST-PUR-DATEPIC X(6) VALUE SPACES.
    03 CUSTOMER-BAL PIC S9(9) COMP VALUE 0.
    03 COMMENTS     PIC X(60) VALUE SPACES.
.
.
01 CUST-KEY        PIC X(16) VALUE "CUSTOMER-NAME;".
01 STATE-KEY       PIC X(16) VALUE "STATE;".
01 CITY-KEY        PIC X(16) VALUE "CITY;".

01 DBSETS.
    03 DBS-SET     PIC X(16) VALUE "CUSTOMERS;".
.
.
01 KEY-VALUE.
    03 KEYWORDS-IN  PIC X(72) VALUE SPACES.
    03 TERMINATOR   PIC X VALUE ";".

01 WORK-FIELD      PIC X(80) VALUE SPACES.

01 MISCELLANEOUS-FIELDS.
    03 ANSWER       PIC X(02) VALUE SPACES.
    03 ITEM-LIST    PIC X(02) VALUE "@;".
    03 CUST-NO-DISP PIC Z(08)9.
    03 QUAL-COUNT   PIC Z(08)9.
    03 SEPARATOR    PIC X(72).

PROCEDURE DIVISION.

MAIN SECTION.

MOVE ALL "-" TO SEPARATOR.

MOVE 1 TO DBM-MODE.
```

```
CALL "DBOPEN" USING DBN-PATH,
                    DBP-WRITE,
                    DBM-MODE,
                    DB-STATUS.
.
.
PERFORM REQUEST-ARGUMENTS THRU REQUEST-EXIT
 UNTIL KEYWORDS-IN = "EXIT" OR "exit" OR "E" OR "e".


PERFORM CLOSE-DB.
```

The paragraph REQUEST-ARGUMENTS continues to execute until the user types EXIT or exit in response to a prompt for keywords.

```
REQUEST-ARGUMENTS.

 MOVE SPACES TO KEYWORDS-IN.
 DISPLAY " ".
 DISPLAY "Please enter Customer, Contact or Title keywords"
 DISPLAY "or 'E' to exit.".
 DISPLAY " ".
 DISPLAY "You may enter wildcard characters (@,#,?) and".
 DISPLAY "Boolean operators."
 DISPLAY " ".
 ACCEPT KEYWORDS-IN.

 IF KEYWORDS-IN = SPACES
        GO TO REQUEST-CITY.

 IF KEYWORDS-IN = "E" OR "e" OR "EXIT" OR "exit"
        GO TO REQUEST-EXIT.
```

The mode value passed in the next section is the DBFIND keyword retrieval mode (mode 12). The mode could be 1 if the CITY was installed only as a keyword key and not as a sorted key. Still, the specialized DBFIND modes ensure that no ambiguity will result regarding the type of keyed access to use.

```
MOVE 12 TO DBM-MODE.

 CALL "DBFIND" USING DBN-PATH,
                     DBSETS,
                     DBM-MODE,
                     DB-STATUS,
                     CUST-KEY,
                     KEY-VALUE.

* If no records contain the keywords (status 17) or records
* contain the keywords but the Boolean result is a null set
* then give the message that no records qualify.
*
 IF IMAGE-STATUS = 17 OR (IMAGE-STATUS = 0 AND CHAINLEN = 0)
        DISPLAY " "
        DISPLAY "No records qualified for keywords entered"
        DISPLAY " "
        GO TO REQUEST-ARGUMENTS.

 IF IMAGE-STATUS NOT = 0 THEN
        PERFORM ERROR-DISPLAY
        GO TO REQUEST-EXIT.

 MOVE CHAINLEN TO QUAL-COUNT.
 DISPLAY " ".
 DISPLAY QUAL-COUNT "Companies qualified"
 DISPLAY " ".
REQUEST-CITY.
 MOVE SPACES TO KEYWORDS-IN.
 DISPLAY " ".
 DISPLAY "Please enter City, State or Zip Code keywords"
 DISPLAY "or 'E' to exit.".
 DISPLAY " ".
 DISPLAY "You may enter wildcard characters (@,#,?) and".
 DISPLAY "Boolean operators."
 DISPLAY " ".

 ACCEPT KEYWORDS-IN.
 IF KEYWORDS-IN = "E" OR "e" OR "EXIT" OR "exit"
        GO TO REQUEST-EXIT.
```

This section lets the user further qualify records. Note that a leading AND operator is programmatically supplied to apply any records found in the following search to the internal ID list generated by the previous search. If a user enters no keywords, then this search is skipped and the process of getting the records from the ID list (via DBGET) begins.

```
    IF KEYWORDS-IN NOT = SPACES
    STRING "AND ", KEYWORDS-IN DELIMITED BY SIZE
      INTO WORK-FIELD
      MOVE WORK-FIELD TO KEYWORDS-IN
    ELSE
      GO TO SKIP-CITY.

    MOVE 12 TO DBM-MODE.

    CALL "DBFIND" USING DBN-PATH,
                        DBSETS,
                        DBM-MODE,
                        DB-STATUS,
                        STATE-KEY,
                        KEY-VALUE.

     IF IMAGE-STATUS = 17 OR (IMAGE-STATUS = 0 AND CHAINLEN = 0)
      DISPLAY " "
      DISPLAY "No records qualified for keywords entered"
      DISPLAY " "
      GO TO REQUEST-CITY.

    IF IMAGE-STATUS NOT = 0 THEN
      PERFORM ERROR-DISPLAY
      GO TO REQUEST-EXIT.

    MOVE CHAINLEN TO QUAL-COUNT.
      DISPLAY " ".
      DISPLAY QUAL-COUNT " Companies qualified"
      DISPLAY " ".

  SKIP-CITY.

      MOVE "N" TO ANSWER.
      PERFORM GET-REC THROUGH GET-REC-EXIT
      UNTIL ANSWER = "EXIT" OR "exit" OR "E" OR "e".

  REQUEST-EXIT.
    EXIT.
```

The next section of code uses specialized DBGET modes (modes 25 and 26) to read forward and backward through the ID list to get the records associated with the IDs. Entering N retrieves the next record for display, entering P retrieves the previous record.

```
GET-REC.

* Mode 25 is a forward chained read. Mode 26 is a backward
* chained read.
 IF ANSWER = "N" OR "n"
         MOVE 25 TO DBM-MODE
 ELSE
   IF ANSWER = "P" OR "p"
          MOVE 26 TO DBM-MODE
 ELSE
   DISPLAY " "
   DISPLAY "INVALID REPLY! "
   DISPLAY " ".

 CALL "DBGET" USING DBN-PATH,
                    DBSETS,
                    DBM-MODE,
                    DB-STATUS,
                    ITEM-LIST,
                    DB-CUSTOMERS,
                    DUMMY.

 IF IMAGE-STATUS NOT = 0 THEN
          PERFORM ERROR-DISPLAY
 ELSE
          PERFORM DISPLAY-REC.

 MOVE SPACES TO ANSWER.

   DISPLAY " ".
   DISPLAY "Please enter an action: ".
   DISPLAY " ".
   DISPLAY " 'N' to see the next record".
   DISPLAY " 'P' to see the previous record".
   DISPLAY " 'E' to exit".
   DISPLAY " ".

 ACCEPT ANSWER.
GET-REC-EXIT.
 EXIT.
```

```
DISPLAY-REC.

 MOVE CUSTOMER-NO TO CUST-NO-DISP.

 DISPLAY " ".
 DISPLAY SEPARATOR.
 DISPLAY "CUSTOMER NO : ", CUST-NO-DISP.
 DISPLAY "CUSTOMER NAME: ", CUSTOMER-NAME.
 DISPLAY "CONTACT : ", CONTACT.
 DISPLAY "TITLE : ", TITLE.
 DISPLAY "CITY : ", CITY.
 DISPLAY "STATE : ", STATE.
 DISPLAY "ZIP CODE : ", ZIP.
 DISPLAY SEPARATOR.

EXPLAIN-DISPLAY.

 DISPLAY " ".
 CALL "DBEXPLAIN" USING DB-STATUS.
 DISPLAY " ".
 DISPLAY "PRESS RETURN TO CONTINUE.".
 ACCEPT ANSWER.

ERROR-DISPLAY.

 DISPLAY " ".
 MOVE SPACES TO ERROR-MSG.
 CALL "DBERROR" USING DB-STATUS, ERROR-MSG, MESSAGE-LEN.
 DISPLAY " ".
 DISPLAY ERROR-MSG.
 DISPLAY " ".
 DISPLAY "PRESS RETURN TO CONTINUE.".
 ACCEPT ANSWER.

CLOSE-DB.

 MOVE 1 TO DBM-MODE.
 CALL "DBCLOSE" USING DBN-PATH,
                      DBSETS,
                      DBM-MODE,
                      DB-STATUS.
 IF IMAGE-STATUS NOT = 0 THEN PERFORM ERROR-DISPLAY.
 STOP RUN.
```

Interfacing with 4GLs

The advanced retrieval capabilities of OMNIDEX can be implemented easily with fourth generation languages (4GLs) such as:

- Speedex from Speedware Corp.
- DATA Express from IMACS Corporation
- Insight from Unison Software Inc.
- Synergist from Gateway Systems Corporation
- Visimage from Ares Corporation
- QUICK from Cognos Corporation's PowerHouse system
- TRANSACT/3000 from Hewlett-Packard's Rapid system
- PROTOS from PROTOS Software Corporation

Speedware Corp.'s Speedex has integrated the OMNIDEX intrinsics, so Speedware products automatically interface with OMNIDEX and require no additional programming.

Unison Software's Insight and Gateway's Synergist have embedded the OMNIDEX intrinsics, so they automatically interface with OMNIDEX and require no additional programming. Synergist also can call the OMNIDEX intrinsics from a personal computer.

Other 4GLs can also be used with OMNIDEX. Documentation on QUICK, TRANSACT/3000 or PROTOS can be obtained from DISC. Call your DISC sales representative or the DISC Response Center for additional documentation.

Glossary

Back out feature	the ability to undo an ODXFIND keyword search and to restore the previous internal ID list
Base ID	a value returned by TurboIMAGE to the ***base*** parameter when the database's name is passed with two trailing blanks
Boolean operators	AND (,), OR (+), and NOT (,-) operators used to combine keywords during an OMNIDEX retrieval
Call Conversion library	procedures that automatically intercept calls to TurboIMAGE intrinsics and convert them into calls to OMNIDEX intrinsics
Calling errors	syntax errors that cause programs to fail. For example, trying to read an entry in Sorted- key sequence by an item that is not an IMSAM key would yield a calling error.
Compatibility mode	a term describing programs and library routines created on Classic HP computers that are run on the MPE XL or MPE/iX operating system
Compound item	a TurboIMAGE item comprised of more than one element (like a 5X50 type field)
Composite keys	a logical Multiple or Sorted key comprising several fields or parts of fields
DBMGR	the DISC database management utility
Data set	a collection of data entries where each entry contains a group of data items. Also called a *table*.

Data type discrepancy	condition in which data stored in a particular field does not match the TurboIMAGE data type defined for that field
Domain	a single data set installed with Multiple keys, or an association of several data sets installed with Multiple keys, and linked during installation. Also called an *OMNIDEX domain*. See *SI domain* and *DR domain*.
DR domain	an OMNIDEX domain (see *Domain*) that consists of one detail data set that has not been linked to a master data set
Exceptional conditions	errors that occur after an intrinsic is called; these errors prevent the call from executing successfully. Trying to read in a Sorted key sequence beyond the last key value would yield an exceptional condition error.
Executable library	HPPA native mode library of callable routines. In native mode versions of OMNIDEX, the OMNIDEX intrinsics, and the call conversion procedure segments, are located in executable libraries (XLs).
Excluded words file	see *Excluded words list*
Excluded words	words specified through OmniUtil for exclusion from the OMNIDEX indexes
Excluded words list	an ASCII file that contains all of the excluded words
General intrinsics	OMNIDEX intrinsics that are used to perform locks, updates, and other general functions
Generic search or Generic retrieval	an OMNIDEX search by a partially specified key followed by an at sign (@)
Grouping	an OMNIDEX key option that treats several Multiple keys as one logical entity
HPPA	Hewlett Packard Precision Architecture. A RISC architecture used in series 9*nn* computers.
IMSAM	the **IM**AGE **S**equential **A**ccess **M**ethod, which provides sorted-sequential and partial-key access on TurboIMAGE databases
IMSAM key	see *Sorted key*

Index-only mode	term applied to high-performance retrievals and updates that affect the key value only. Discrete mode IMSAM retrievals return only the key values stored in the index sets without returning a record.
Internal ID list	the most current list of record identifiers qualified by an ODXFIND
Item list	the items passed through the ***list*** parameter
Key	a field that is used to select records from a data set
Key options	enhancements to OMNIDEX keys that are specified during installation to enable certain retrieval features (like Grouping) or update features (like Batch Indexing)
Keyed retrieval	retrieval by an OMNIDEX or TurboIMAGE key field
Keyword	a word or number that is indexed for a Multiple key and stored in the OMNIDEX indexes
Keyword retrieval	refers to an OMNIDEX retrieval of record identifiers by any keyword value or combination of keyword values against a Multiple key
Logical link	used in relation to Multifind, when data in two different fields in two different data sets serve to create a correspondence between those records
Mode	an intrinsic parameter that specifies what type of action is to be performed
Multifind	a retrieval feature that supports keyword searches across domains using field values from previously qualified records as keyword arguments against a table in a different domain
Multiple key	a field specified for OMNIDEX keyword retrieval
Multiple key access	the ability of OMNIDEX to qualify records based on arguments entered against several keys
Multi-RIN (MR) capability	a capability used to lock several data resources simultaneously. This capability is used by programs that update or lock OMNIDEX databases, data sets, or items.

Native mode	a term attributed to programs and library routines that can be directly executed on a Series 9*nn* HPPA computer
Native Language Support	the collating of 8-bit extended character sets (Arabic8, Greek8, Kana8, Roman8, Turkish8) to ASCII as handled by OMNIDEX and the HP 3000
Next free ID	an internally maintained value that indicates the next assignable integer SI value. May be an ID previously used by a deleted record.
Next unused ID	the ID value that is one higher than the highest used value
Noise words	words that occur frequently or are useless for retrieval (like "the" or "therefore")
OMNIDEX condition word	word 11 of the ***status*** array, which contains an error code if a call to an OMNIDEX intrinsic failed
OMNIDEX detail	a detail set linked to an OMNIDEX master by an OMNIDEX SI (search item)
OMNIDEX domain	see *Domain*
OMNIDEX error indicator	the word of the *status* array that indicates whether an error has occurred. This word differs between default (active) error handling and Passive Error Handling.
OMNIDEX ID	a binary integer (I2, J2, or K2) value that is used to identify records during keyword searches (for Multiple keys)
OMNIDEX Intrinsic Interface	OMNIDEX intrinsics that provide retrieval access to Multiple and Sorted key indexes and maintenance to OMNIDEX indexes
OMNIDEX master	a master data set that contains one or more fields specified for OMNIDEX keyword retrieval, or a master that is linked by the OMNIDEX SI to a detail that contains one or more Multiple keys
OMNIDEX rootfile	the index file that contains information about the database structure, including which fields have been specified for OMNIDEX retrieval

OMNIDEX SI	the TurboIMAGE search item field in an OMNIDEX master. It may also be the OMNIDEX ID for the set.
One-to-many relationship	refers to data entities where one entity corresponds to many other entities, as in the case of a search item value to many detail records
One-to-one relationship	refers to data entities where one entity corresponds only to one other entity, as in the case of a search item value to a master record
Operator	any special character or token used in OMNIDEX retrievals. For example, the Multifind operator (&).
Parameter	a variable value that can be specified when issuing a command or calling an intrinsic
Parsing	separating a character string into individual keywords
Partial-key retrieval	refers to a Sorted key retrieval where a partially specified search argument is used to retrieve one or more records
Qualifying count	the number of search items or TurboIMAGE record numbers that qualify in an OMNIDEX keyword retrieval
Record complex	an affiliation of records in an SI domain that all contain the same OMNIDEX SI value. Also refers to keys that qualify record complexes (Multiple keys in master data sets, and Multiple keys installed with the Record Complex option).
Record number	an internally maintained value associated with the position in a detail file of individual detail records
Record specific	refers to Multiple keys or keyword retrievals that qualify individual detail records, as opposed to record complex. Non-Record Complex keys in details are record specific.
Relational operator	the greater than (>), greater than or equal to (>=), less than (<), less than or equal to (<=) and equals (=) operators used to specify a relational retrieval
Relative record number	see *Record number*
Retrieval intrinsics	intrinsics that are used to search for or retrieve records

Root file	a file that contains information about the database structure. In OMNIDEX, the OMNIDEX root file contains information about where keys are installed.
SI	commonly-used abbreviation for **S**earch **I**tem
SI domain	an affiliation of one master set and one or more detail sets, linked at installation, whose keyword values are all indexed in the same indexes
SL	commonly-used abbreviation for **s**egmented **l**ibrary
Samelist operator	the asterisk (*), which tells OMNIDEX that you wish to further qualify the list of IDs
Search item (SI)	an TurboIMAGE field within a data set that is used for calculated access in master sets and chained access in detail sets
Segmented library (SL)	a collection of procedure segments. In version 3.0 of OMNIDEX, the OMNIDEX Intrinsic Switch Stub segments, are located in a segmented library (SL.PUB.DISC).
Serial read	a sequential read of records in a data set that is performed one block of records at a time. For a master set, a serial read ends at the end of file. For a detail set, it ends at the high-water mark.
Special character	a character that is parsed by OMNIDEX before being indexed. Likewise, a character that is interpreted by OMNIDEX to have a special meaning.
Sorted key	a field or composite key specified for sorted access. Sorted key retrievals return records sorted by key values.
Sorted access	the ability to retrieve records in sorted order through the use of a Sorted key
Source	in Multifind usage refers to the item, set or database that is supplying the data values used as search arguments
Split retrieval	refers to a keyword search on a record specific key that immediately precedes or immediately follows a search on a record complex key

Subfield	in TurboIMAGE, the individual elements that comprise an array. For example, each X50 element is a subfield in a 5X50 array.
Switch stub	a routine that allows Compatibility Mode programs to call OMNIDEX version 3.0 Native Mode intrinsics
TPI-enabled	refers to a database whose OMNIDEX indexes are accessible to the Standard Interface to Third Party Indexing
Table	see *Data set*
Target	when used in reference to Multifind, refers to the domain, data set or field in which a list of keyword values is to be further qualified. When used in reference to DISC utilities, it refers to a data entity on which a specified operation is to be performed.
Textual data	data containing words, phrases, and sentences (as opposed to numeric, or other fixed data)
TurboIMAGE condition word	word 1 of the ***status*** array, which contains an error code if a call to an TurboIMAGE intrinsic failed
USL	**U**ser-**S**egmented **L**ibrary. It contains Relocatable Binary Modules that can be added to an SL.
Upshifting	shifting of alphabetic characters to upper case in the OMNIDEX index sets
Wildcard	any character that is used to imply a partial search argument. The at sign (@) is used as a wildcard in generic retrievals.
XL	see *Executable library*

Index

Symbols

Pound sign
  as wildcard 2-61, 2-109
%DATE function 2-59
& Ampersand
  in Multifind 2-66
( ) Parentheses
  in Boolean expressions 2-58, 2-64
* Asterisk
  reloading the ID list 2-69
< Left angle bracket
  to undo keyword search 2-68
? Question mark
  as wildcard 2-61, 2-109
@ At sign
  as *argument* terminator 2-108
  as wildcard 2-109
[Ctrl]-Y
  ODXFIND 2-55

Numerics

3GLs 1-12, 3-40
4GLs 3-60

A

Account capabilities 3-39
Active Error Handling 3-5
Applications
  3GLs 3-40
  4GLs 3-60
  adding capabilities 3-38
  choosing an interface 2-10
  compatibility mode 1-17, 3-36
  developing 1-11
  migrating to other platforms 2-2
  OMNIDEX searches in IMAGE 2-106
  testing 3-33
argument parameter
  data type 2-105
  DBFIND binary range 2-110
  in sorted access 2-13
  terminating for DBFIND 2-108
  values vs. operators 2-96
Arguments
  in Boolean operations 2-62
ASCII fields
  range restrictions 2-65
ASKDATE format 2-60
  data type 2-60

B

Back-out feature
  OMNIDEX Intrinsic Interface 2-68
  TPI interface 2-102
Base ID 2-44
base parameter
  OMNIDEX Intrinsic Interface 2-44
Batch applications
  the internal ID list 2-100
Binary data
  sorted range retrievals 2-110
Boolean operations
  coded into programs 3-57
  DBFIND mode 1 2-107, 2-109
  DBFIND TPI modes 2-114
  ODXFIND 2-62
  order of precedence 2-58, 2-64, 2-112
  parenthetical nesting 2-58
  reloading the ID list 2-69

C

Call Conversion library 1-17, 3-2, 3-28, 3-34
see also XL.PUB.DISC
adding to existing XL 3-37
effect on updates 3-28
for testing applications 3-33
Calling errors
status values 3-7
Chain boundaries
for sorted access 2-120
Chain count
sorted retrieval 2-111
Chained reads
skipping sorted records 2-120
COBOL examples 3-40, 3-49
Compatibility mode 1-17
Composite keyword keys
identifying components 2-84
Composite sorted keys
data type 2-105, 2-110
Condition word
IMAGE 3-6
OMNIDEX 3-7
Conventions
used in this guide ix
Critical item update 2-51

D

Data type
DBFIND *item* parameter 2-108
of composite keys 2-105, 2-110
of DBFIND *argument* 2-105
Database
see also Exclusive access
enabling for TPI 3-27
mode 3 item list 2-50
opening 3-2
opening in IMAGE 3-27
opening in OMNIDEX 2-44, 3-2
TPI-disabled 3-34, 3-35
TPI-enabled 2-9, 3-34, 3-35
DataView 1-15
Dates
see also ASKDATE format
see also JDATE format
see also PHDATE format
converting arguments 2-59
partial arguments 2-60
reordering fields 2-60
DBCONTROL 2-99
DBFIND 2-101
IMAGE modes 2-101
item parameter and OMNIDEX 2-107
keyword searches 2-112, 3-54
parameters 2-101
sorted access 3-49
DBGET
after DBFIND mode 1 2-121
IMAGE reads 2-117
OMNIDEX reads 2-118
parameters 2-117
reading ID lists 2-121, 3-54
reading sorted chains 3-49
reading sorted key chains 2-120
DBICLOSE
parameters 2-16
DBIDELETE
indexing errors 2-19
parameters 2-18
DBIERROR
parameters 2-20
DBIEXPLAIN
parameters 2-22
DBIFIND
COBOL program examples 3-41
errors 2-27
IMAGE modes 2-23
index-only mode 2-26
mode options 2-24
parameters 2-23
relational modes 2-24
sequential modes 2-24

DBIGET
COBOL program examples 3-43
errors 2-32
IMAGE modes 2-28
index-only mode 2-31
mode options 2-29
normal mode 2-31
parameters 2-28
relational modes 2-29
sequential modes 2-29
DBIINFO
errors 2-41
parameters 2-34
DBILOCK
parameters 2-42
DBINFO
calling errors 2-128
parameters 2-123
DBIOPEN
exceptional conditions 2-45
opening databases 3-2
parameters 2-44
DBIPUT
calling errors 2-48
exceptional conditions 2-48
indexing errors 2-47
mode options 2-46
parameters 2-46
DBIUNLOCK
parameters 2-49
DBIUPDATE
parameters 2-50
Detail Record indexed domains
effect on DBIPUT item list 2-46
effect on ODXTRANSFER 2-89, 2-91
finding with ODXINFO 2-83
OMNIDEX ID 2-121
Detail set
linked 2-62, 3-16
returning SIs 2-62
unlinked 2-73, 3-16
Discrete mode
updates 3-4
dset parameter
DBIGET vs. DBGET 2-5

E

Errors
3000 series 3-8
see also Calling errors
see also Exceptional conditions
calling vs. exceptional conditions 2-21
condition words 3-5
DBFIND 2-116
DBGET 2-122
DBIDELETE 2-19
DBIFIND 2-27
DBIGET 2-33
DBIINFO 2-41
DBINFO 2-128
default for OMNIDEX Intrinsic Interface 2-6
handling 2-6, 2-22
help with 3-8
IMAGE 3-28
IMAGE condition word 3-6
IMAGE in OMNIDEX 3-6
IMSAM and IMAGE 2-122
indexing 2-47
ODXFIND 2-69
ODXGET 2-77
ODXGETWORD 2-79
ODXINFO 2-85
ODXPRINT 2-87
ODXTRANSFER 2-91
ODXVIEW 2-94
OMNIDEX 3-4
OMNIDEX condition word 3-7
status values for 3-5
summary of OMNIDEX 3-8
summary of TPI 3-28
TPI error codes 3-8

Exceptional conditions
DBFIND 2-116
status values 3-7

F

Fields
reordering dates 2-60
transferring data from 2-88
Files
appending to 2-88
creating for Multifind 2-89, 2-90
printing 2-86
saving internal IDs 2-90
using in Multifind 2-67, 3-20
viewing 2-92
FUTIL
changing prrograms' capabilities 3-38

G

General intrinsics 2-4, 3-35
Generic arguments
dates 2-60
in sorted retrievals 2-13
length specification 2-25, 2-30
longer than 100 bytes 2-25
ranges 2-65
Grouping
identifying keys in group 2-82

I

ID list preservation
disabling 2-55, 2-100
in TPI Interface 2-99
IMAGE
DBGET reads 2-117
mode values 2-5
Standard Interface to TPI 2-95
version information 3-34
IMAGE condition word 3-6
IMAGE-only mode 2-5, 2-12
DBIDELETE 2-18
DBIPUT 2-46
updates 3-4
IMSAM
see Sorted access
see Sorted keys
sorted sequential retrievals 3-23
Indexes
automatic updating 2-9
when no records qualify 2-68
Index-only mode 2-5, 2-12
DBIFIND 2-23, 2-26
DBIPUT 2-46
Internal ID list 2-73, 2-96, 2-121
see also Back-out feature
batch applications 2-100
circular 2-74
compressing 2-62
defined 2-68
end of chain condition 2-121
end of list condition 2-73
linear 2-75
preservation 2-99
reloading 2-69
saving 2-89
when no records qualify 2-99
Intrinsic Switch Stubs 1-17, 3-36
Intrinsics
function 2-2
general functions 2-4
IMAGE 2-7, 2-95
IMAGE vs. OMNIDEX 1-12, 2-3
index maintenance 2-4
keyword access 2-3, 2-52
OMNIDEX 1-16
parameters in OMNIDEX 2-4
retrieval 2-10
sorted access 2-3, 2-13
update 2-9

see also individual listing
Intrinsics library
defined 1-16
installing 3-2
OMNIDEX 1-16, 1-17
Item list 2-46
effect of ODXTRANSFER mode 101 on 2-90

J

JDATE format 2-60
data type 2-60

K

Key options
identifying 2-83
Keys
access ambiguities 2-109
keyword 1-2
referenced in DBFIND *item* 2-107
sorted 1-5
Keyword access
intrinsics 1-16
retrieving records (DBGET) 2-118
retrieving records (ODXGET) 2-74, 2-121
Keyword access intrinsics 2-3
Keyword keys 1-2
getting information about 2-80
Keyword search 2-52
Boolean operations 2-62
defined 1-4
enhanced argument parsing 2-58
interactive vs. batch 2-68, 2-100
interrupting 2-55
intrinsics 2-52
intrinsics called 3-13
keywords only 2-78
literal operators 2-58
master set 2-73, 2-121
OMNIDEX intrinsics 2-3
on a detail 3-15
on a master 3-13
operations supported 2-53
qualifying count for details 2-56
Record Complex (RC) option 2-73, 2-121
Record Complex keys 3-17
record retrieval (DBGET) 2-118
record retrieval (ODXGET) 2-74, 2-121
record specific 3-16
relational operations 2-65
saving an ID list 2-89
through DBFIND 2-112
TPI Interface 3-29
undoing 2-68, 2-102
unlinked detail 3-16
Keyword-only search 2-62, 2-78, 3-19
retrieving keywords 2-78
Keywords
indexing 2-52
keywords parameter 2-62
length and terminator 2-57

L

Libraries
combining XLs 3-37
Link Editor from HP 3-37
Locking
Multi-RIN (MR) capability 3-38
OMNIDEX Intrinsic Interface 2-42

M

Master data sets
Transparent ID 2-50
Mode options 2-11
defined 2-5
MODE parameter
see also Modes
mode parameter 2-13
generic arguments 2-25
TPI Interface values 2-8

Modes
see also Mode options
defined 2-5
IMAGE-only 2-5
index-only 2-5
normal 2-5
Relational 2-111
MPE iX 2-2, 3-34
Multifind 2-66, 3-19
creating a file 2-90
intrinsics called 3-19
Multiple access
see also Keyword search
Multi-RIN (MR) capability 2-43, 3-38, 3-39

N

Normal mode 2-5, 2-11
updates 3-3

O

ODXFIND
calling errors 2-70
COBOL program examples 3-44
exceptional conditions 2-69
function 2-52
mode 30 2-62
mode options 2-55
modes 2-54
modes 10 and 11 2-62
modes 3 and 5 2-58
parameters 2-54
relational operations 2-65
ODXGET
calling errors 2-77
COBOL program examples 3-45
exceptional conditions 2-77
internal ID list variations 2-73, 2-121
mode options 2-72
parameters 2-71
ODXGETWORD
calling errors 2-79
COBOL program examples 3-46
exceptional conditions 2-79
parameters 2-78
ODXINFO
calling errors 2-85
parameters 2-80
ODXPRINT
calling errors 2-87
parameters 2-86
ODXTRANSFER
calling errors 2-91
COBOL program examples 3-47
mode options 2-88
parameters 2-88
ODXVIEW
exceptional conditions 2-94
parameters 2-92
OMNIDEX
3GLs 3-40
4GLs 3-60
application development 1-9
benefits and features 1-7
components of 1-14
DBGET reads 2-118
defined 1-2
error handling 2-6
flexibility 1-8
intrinsic parameters 2-4
intrinsics 1-12
procedure libraries 1-11
reliability 1-10
speed of access 1-7
technical support x
OMNIDEX condition word 2-6, 3-5, 3-7
OMNIDEX domains
see OMNIDEX masters
OMNIDEX IDs 2-96
assigning through DBIPUT 2-47
determining type 2-83
next available ID 2-84
OMNIDEX Intrinsic Interface 1-9, 3-2
features 2-10

overview 2-2
OMNIDEX masters
identifying for database 2-82
OmniUtil 1-14
Opening databases
OMNIDEX Intrinsic Interface 2-44, 3-2
TPI Interface 3-27

P

Partial key retrieval
see Generic arguments
Passive error handling 3-9
TPI-enabled databases 3-9
Pattern matching 2-61, 2-109
relational operations 2-61
PHDATE format 2-60
data type 2-60
Precedence of operations
ODXFIND 2-58
Product support x
Programs
COBOL examples 3-40, 3-49
development effort 1-13
testing 3-33
PROTOS 1-12

Q

Qualifying count
see also Chain count
converting to Record Complex 3-16
DBFIND on RC keys 2-113
default keys in details 3-16
intrinsics called 3-19
keyword-only search 2-78
ODXFIND 2-56
sorted retrievals 2-111
status words 2-56
QUICK 1-12

R

Range operations
ASCII restrictions 2-65
DBFIND binary sorted 2-110
in DBFIND 2-113
ODXFIND 2-64
sorted ASCII 2-110
RAPID 1-12
Record Complex (RC) option 3-17
effect on multi-key searches 2-113
Record complexes 2-55
ODXGET 2-73, 2-121
qualifying count 2-56
Record Complex (RC) option 2-112
Record specific keys 2-73, 2-88, 2-90, 2-121, 3-16
Records
adding 2-46
Relational modes
DBFIND 2-111
DBIFIND 2-24
DBIGET 2-29
Relational operations
DBFIND binary sorted 2-111
DBFIND mode 1 2-110
DBIFIND 2-24
DBIGET 2-29
generic arguments 2-14
in DBFIND 2-115
ODXFIND 2-65
pattern matching 2-61, 2-110
sorted keys 2-13, 2-97
sorted keys DBFIND 2-102
Relational positioning modes 2-102
Retrievals
see also Discrete Mode
after keyword searches 2-121
DBFIND argument terminators 2-108
DBFIND mode 1 2-109

IMAGE interface 2-96
keyword 3-13
of chain heads 2-24
sorted 2-120
sorted key 3-22

S

Search items
in DBFIND *item* 2-107
Search values
see Arguments
Segmenter from HP 3-37
Semicolon
as terminator 2-57, 2-92, 2-108
SL.PUB.DISC 3-36
Sorted access
chain count 2-111
DBFIND 2-104
finding chain heads 3-24
index-only mode 2-32, 3-26
OMNIDEX Intrinsic Interface 2-13
OMNIDEX intrinsics 2-3
partial key arguments 2-97
partial-key retrieval 3-22
relational operators 2-97
resetting record pointer 2-120
retrieving records (DBGET) 2-118
to detail chains 2-23
tokens in DBFIND 2-109
TPI Interface 3-31
Sorted access intrinsics 2-3
Sorted keys 1-5
verifying presence and type 2-34
Sorting records 1-5
status array 2-6
IMAGE vs. OMNIDEX 2-5
in OMNIDEX 3-6
ODXFIND values 2-56
word values 2-6

T

Technical support x
Third Party Indexing Products
see TPI Interface
TPI Interface 2-95, 3-27
COBOL program examples 3-49
defined 1-10
error handling 2-9, 3-28
features 2-8
overview 2-7
passive error handling 3-9
role of MPE iX 2-2
version information 3-34
Transparent domains
DBIPUT item list 2-46
effect on ODXTRANSFER 2-91
finding with ODXINFO 2-83

U

Updates
adding records 2-46
IMAGE-only mode 2-12
index-only mode 2-12
normal mode 2-11
of indexes 2-9
of indexes (automatic) 2-50
OMNIDEX intrinsics 3-3
real-time indexing for TPI 3-28
TPI applications 3-28
Utilities 1-14

V

V/3000 2-55

W

Wild cards
see Pattern matching

X

XL.PUB.DISC 3-2, 3-27, 3-28, 3-34
XL.PUB.SYS 1-11