SUPERDEX

Enhancing your database

SUPERDEX

User Manual

Version 3.1

All updates to or derivatives of the SUPERDEX™ computer software provided herein are copyrighted and may not be copied except for archive purposes, to replace a defective copy, or for program error verification by Licensee. Copyrighted material may not becopied onto any media (e.g. magnetic tape, paper tape, disc memory cartridges, read-only memory, etc.) for any other purposes. The authorization to duplicate copyrighted materials hereunder shall not be construed to grant the Licensee or Licensee's customer the right to use copyrighted SUPERDEX material in any manner other than which is provided in this agreement or otherwise approved in writing by Bradmark Techologies, Inc..

Released March, 1992

About this manual

In writing this manual, we have assumed that you have working knowledge, although not internal knowledge, of IMAGE and the HP3000.

All references to IMAGE in this manual and throughout the SUPERDEX package also apply to TurboIMAGE and TurboIMAGE/XL unless otherwise noted.

This manual is arranged in the following format:

Section 1 provides an *Overview* of the SUPERDEX package, its capabilities, and benefits. It also describes how SUPERDEX works and how it maintains compatibility with IMAGE and its facilities.

Section 2 overviews the various *Access methods* available in SUPERDEX for qualifying and retrieving entries in IMAGE databases, with one chapter per method.

Section 3 describes the procedures used in *Configuration/Establishing SI-indices* in SUPERDEX for use with your databases to provide quick retrieval of data entries.

Section 4 discusses the various methods utilized in *Programming* with SUPERDEX to add, update, delete, qualify, and retrieve entries in SUPERDEX, and gives examples of each.

Section 5 describes the *Intrinsics* provided with SUPERDEX as enhancements to the IMAGE intrinsics, as well as new SUPERDEX intrinsics.

Section 6 discusses the various *Maintenance* considerations for SUPERDEX'ed databases, as well as the use of various *Utilities* to access and maintain them.

Section 7 deccribes *SuperSELECT*, which can be used to override a serial read with a SUPERDEX qualification and retrieval.

Appendix A contains fragments of *Program examples* written in COBOL utilizing SUPERDEX.

Appendix B examines various *Internal structures* used for SUPERDEX, including the method for calculating the capacity of the SI-dataset.

Appendix C documents *Maximum limits* that are imposed for SUPERDEX configuration.

Appendix D lists *Error and exceptional conditions* for SUPERDEX intrinsics, utilities, and programs with their meanings and recommended actions.

Following Appendix D are several **Supplements** for different products and languages.

Finally, the **Index** is a complete index to the manual.

Table of contents

Overview

SUPERDEX is not a database management system, nor a programming language. It is realized as a natural extension to IMAGE, picking up where IMAGE leaves off. SUPERDEX automatically creates and manages new B-tree indices in your databases and provides enhanced IMAGE-compatible intrinsics used by your programs automatically.

SUPERDEX has been designed and implemented to provide the most power and flexibility with the least amount of effort. SUPERDEX is so simple, it requires little training and takes only a few minutes to configure and minor program modifications to implement. Many SUPERDEX capabilities are accessible with no program modifications at all.

This section previews the SUPERDEX package, its capabilities, and its benefits.

Charter 1   Why SUPERDEX?
Description   explains the basic reason for SUPERDEX and it's capabilities.

Chapter 2   Enhancements
Description   is a description of the latest *enhancements* in SUPERDEX Version 3.1.

Chapter 3   Functionality
Description   gives a brief description of the features that provide SUPERDEX's *Functionality* and a simple example of each.

Chapter 4   Concepts
Description   explains the main *Concepts* of SUPERDEX and how they are used.

Chapter 5   Terminology
Description   defines the *Terminology* used throughout SUPERDEX to identify its features and capabilities.

Chapter 6   Access Principles
Description   overviews the *Access principles* within SUPERDEX for adding, updating, deleting, and retrieving entries.

Chapter 7   Compatibility
Description   reviews *compatibility* issues, including compatibility with IMAGE intrinsics, the *status* array, and transaction logging. Additionally, the impact on existing application programs, third- and fourth-generation languages, QUERY and other utilities is discussed. Finally, issues involving locking, program capabilities, and stack requirements are examined.

Why SUPERDEX ?

Perhaps the best way to understand what SUPERDEX is all about is to understand why it was created, why it was implemented in the way it was, and why we believe you will find it to be a simple, straightforward method for achieving faster, more flexible access to your IMAGE, TurboIMAGE, and TurboIMAGE/XL databases.

IMAGE, although a very functional and powerful database management system, lacks certain obviously-needed capabilities, such as:

- multiple keys in both master and detail datasets
- concatenated keys containing multiple fields
- sorted sequential retrieval
- automatic keywording and keyword retrieval
- generic and partial-key retrieval
- approximate match retrieval
- greater-than, less-than, and range retrieval
- grouping of functionally-equivalent fields
- relational access using multiple criteria
- relational access across multiple fields, datasets, and databases

SUPERDEX provides these desirable capabilities, and several others.

In IMAGE, for example, a master set indexes entries in a detail set, but there is no IMAGE structure that indexes a master set. To locate a master entry, you must specify the exact search field (key) value. To get around this limitation and permit entries to be located by multiple keys, many IMAGE databases are designed such that master-oriented information (entities such as customers, vendors, and parts) are placed in detail sets instead of master sets and indexed via automatic master sets. This leads to cumbersome, inefficient database structures, yet fundamental operations such as partial-key retrieval still cannot be performed without serially reading the dataset.

One fundamental requirement in designing SUPERDEX was to provide complete flexibility in searching for entries by allowing any field in any master or detail dataset to be used as a key. SUPERDEX permits you to designate any field--even every field--in master and detail datasets as a key.

Another requirement was to be able to specify a partial or generic key value in searching for entries. SUPERDEX can locate entries by a partial or generic key, multiple keys, or a range of keys. It can even find entries by any word contained in a key, which is called keyword retrieval.

SUPERDEX provides these capabilities for master and detail sets in the same way, so the same code that is written for detail datasets will work on master datasets, and vice-versa. In fact, ease of integration was one of SUPERDEX's primary requirements.

Another requirement was to minimize space and resources and impact existing structures as little as possible. So, SUPERDEX requires only a single dataset with a single field in your database, in which it maintains special B-tree index structures. Alternately, SUPERDEX's indices can reside in a separate database. SUPERDEX manages these indices automatically using standard IMAGE intrinsics, to remain entirely within and compatible with IMAGE facilities such as transaction logging, unlike other indexing structures such as KSAM. These B-tree structures may be established, deleted, and reconstructed quickly on-the-fly, allowing very flexible indexing schemes like those inherent in relational databases.

Another IMAGE limitation is the inability to relate a master set to another master set. SUPERDEX permits logical master-master connectivity, and moreover, connectivity between multiple keys in a dataset, between any logically related datasets, and even across multiple databases.

The overall intent in creating SUPERDEX was to make it look and feel just like IMAGE--perhaps the next logical step for IMAGE--with fully-compatible enhanced intrinsics that provide additional functionality with the same names, parameters, and calling sequences. In SUPERDEX, all access capabilities are available through DBFIND *mode* 1 to find entries and DBGET *mode* 5 and 6 to retrieve them. And, to make matters as simple and straightforward as possible, all selection criteria may be specified in the *argument* for DBFIND *mode* 1, permitting generic retrieval code to be written and the type of and scope of the retrieval specified by the user.

SUPERDEX's B-tree indices are maintained automatically by DBPUT, DBDELETE, and DBUPDATE. All status information is returned in the standard IMAGE *status* array, and locking may be done automatically, further easing integration.

And most important of all, SUPERDEX was designed to be the fastest method for retrieving entries in an IMAGE database. Lookups that would take minutes in IMAGE are done in seconds by SUPERDEX.

We know you will quickly see the areas in which SUPERDEX improves IMAGE, and appreciate the efforts that have been taken to make SUPERDEX as easy for you as its speed and power are to your users.

Enhancements

Version 3.1 has several enhancements that are briefly described here.

SUPERDEX Menu Program

The SUPERDEX menu program provides a full screen system that can access all of the programs and utilities in the SUPERDEX product. This menu also provides a user-friendly, menu-driven process that allows the user to maintain SUPERDEX paths.

SuperSELECT

This new program is used to intercept an existing program that reads the database serially. It will force the serial read to be replaced with SUPERDEX qualifying and reading. It can be used to greatly increase the speed of serial reads which also do qualifications.

The are several different modes of execution, including the ability to build entry screens for user-friendly prompts.

SIREPAIR

SIREPAIR is a program that can be used to repair indices that are not aligned with the data. This situation occurs when a program that updates the data was executed without the SUPERDEX library. SIREPAIR will determine and report any inconsistencies in a path and will correct those inconsistencies.

SICOUNT

In SUPERDEX Version 3.0, disc usage for indexes was reduced via a compression algorithm that takes into consideration many different aspects of the user data and the database structure. This makes it very difficult to calculate what the exact compression ratio is. SICOUNT has been developed to read the B-tree and display the exact compression status.

SIDRIVER

The new SIDRIVER program replaces the current DBDRIVER.PUB.SUPERDEX. It has several additional features including multiple database access and upshifting.

New Boolean Operators

The words **AND**, **OR**, and **NOT** can now be used in an argument value to perform relational accessing. For example, the user can now enter ˜**SMITH@ OR JONES@;** as a valid search value.

Additionally, the infix notations, **+** (AND), **,** (OR), and **-** (NOT), can be entered as valid operators in an argument. The user can enter ˜**SMITH@,JONES@;**.

New Wild Cards

The **@** can now be used in an argument other than just for a terminators. Up to two (2) **@** signs can be included in an argument. One can be placed at the beginning of an argument, or they can be placed in the middle.

<<A@B@C>> -will retrieve all words that begin with an "A", have a "B" anywhere after the "A", and a "C" anywhere after the "B".

A new wild card for numeric values has also been added. The **#** can be used to float a particular position (similar to the **?** wild card), but will only qualify records that have a numeric value (0 - 9) in the position

AB#J35@ - will retrieve AB3J35P12 and AB5J35T, but will not retrieve ABKJ354.

Paths can now have their starting position defined, along with their lengths defined in bytes. This means a path can now begin in position 5 of the field, and only include the next 3 positions.

SIMAINT Enhancements

Paths can now have their starting positions (referred to as "OFFSET") along with their lengths defined in bytes. This means a path can now begin in position 5 of the field, and only include the next 3 positions.

SIMAINT now has thorough progress reporting. As paths are processed, SIMAINT will display the number of indexes processed, the percentage complete, the elapsed time, and the number of CPU seconds used. The interval for progress reporting defaults to every 1000 records, but can be defined prior to execution.

The number of datasets in a database that can contain SUPERDEX paths has been expanded from 100 to ALL datasets in the database.

The number of grouped items in a path has been increased from 16 to 32.

Path Numbers

Now, as paths are defined, SUPERDEX will assign a path number beginning with the value of ten thousand and one (10,001). This is similar to IMAGE defining and allowing set and item numbers to access data.

DBINFO Mode 312

DBINFO Mode 312 has been modified to return the new path number, along with the starting position of the path. The buffer size of DBINFO mode 312 was not increased. The areas reserved in version 3.0 were utilized. Existing calls will continue to function as before.

SIPATH Enhancements

SIPATH has been enhanced to display the new options available for paths. These include the path number, the subkey starting position and the length of the subkey. Additionally, the IMAGE key information can optionally be displayed.

New Relation Operator

A new option has been added for those situations where several relational DBFINDs are being executed and the relation tables overflow. (See **Section 4:** *Programming*)

SIBASE

This new program will create the additional SI-Index database for storing the SI-indices in a separate database.

SISIZE

This new program allows you to maintain the capacity of the SI-dataset(s).

Functionality

The various capabilities of SUPERDEX are covered here. More complete information about and examples of each feature appears in the *Access methods* section.

Sample applications

SUPERDEX may be used throughout your application systems in different ways to accomplish various operations. Some of the more common uses of SUPERDEX capabilities are listed here:

■ **Customer lookup**
Customers stored in a master dataset need to be accessed by name, contact, phone number, and address. SUPERDEX could search on any field, keyword the contact so that either first or last name or both could be specified, and group together both lines of a two-line address so both are always searched. The customer name could be looked up by a partial or generic key.

■ **Part lookup**
Users can enter partial part descriptions and the program retrieves all that qualify and displays them on the screen with their corresponding part numbers. SUPERDEX could treat the part description as a keyworded field, permitting any word or words within the description to be specified.

■ **Part classification extract**
All part numbers start with a classifying character sequence, and it is necessary retrieve all the parts which start with a certain sequence of characters, so those characters are specified as partial keys.

■ **Mail room**
Everyone in a company has a mail-stop, but not all correspondence indicates it, so the mail clerks enter the addressee's name or partial address and get the mail stop for routing.

■ **Text Management**
Comments and other text must be searchable by any word contained in an 80-character field; SUPERDEX could handle it as a keyworded field.

■ **Library system**
Book titles, authors, and summary information stored in a master set and two related detail sets could be super-grouped together, permitting retrieval by any combination of criteria in a single operation.

Multiple keys in master and detail datasets

IMAGE lets you access a master set by only one field, and a detail set only via its related masters unless time-consuming serial reads are performed, forcing rigid applications and cumbersome database structures. Using simple, IMAGE-compatible techniques, SUPERDEX lets you access any dataset directly by any field, regardless of whether or not it is an IMAGE search field.

For example, a customer entry in a master dataset could be looked up by its customer number, customer name, contact name, or phone number.

Concatenated keys containing multiple fields

SUPERDEX permits multiple fields or truncated fields to be concatenated together and retrieval to be done on the entire concatenated value. This permits very specific lookups to be performed without having to read serially or down a chain to qualify entries that match on multiple fields because all fields may be contained in the key.

For example, a division number, group number, and partial account number could be concatenated together and looked up by the full combined value or any portion of the combined value.

Sorted sequential retrieval

IMAGE returns entries in chronological order, unless sorted paths are used. SUPERDEX returns entries in ascending or descending alphabetical order and, by using concatenated keys, provides more flexibility than sorted paths without the overhead.

For example, a classification number, account number, and date/time stamp could be concatenated together, and entries would be returned in chronological order within each account within each class.

Keyword retrieval

SUPERDEX lets you access entries by any word contained in designated fields. This technique is referred to as keywording.

For example, the entry "BRADMARK TECHNOLOGIES, INC." could be located by specifying either **BRADMARK**, **TECHNOLOGIES**, or **INC**.

Generic and partial key retrieval

IMAGE won't find an entry unless you specify its exact key value. SUPERDEX is far more forgiving: you may specify for any key or keyword a partial value or an embedded value with matchcodes.

For example, **GEN@** would find all the entries that begin with "GEN" and **MA??ER** would find all the entries that begin with "MA" followed by any two characters followed by "ER". **A#J3@** would find all the entries that begin with "A" followed by a single digit (0 - 9), then "J3" followed by any other characters.

Approximate match retrieval

IMAGE cannot not find an entry that does not exist, but SUPERDEX can do the next best thing: find the nearest matching entry.

The alphabetic ordering of indices allows approximate match retrieval: if no matching entry exists, the nearest qualifying entry is returned, permitting a program to start reporting data at any alphabetic location.

Greater-than, less-than, and range retrieval

SUPERDEX is also capable of retrieving all entries that are:

- greater than or equal to a specified value
- less than or equal to a specified value
- not equal to a specified value
- within the range of two values

For example >=1000 would find all the entries with amounts greater than or equal to 1000, <=500 would find all entries with amounts less than or equal to 500, <>10 would find all amounts not equal to 10, and >=A@<=C@ would find all the entries that begin with the letters "A", "B", or "C".

Grouped retrieval

IMAGE can search only one field at a time. SUPERDEX lets you group multiple fields together at configuration time, and automatically searches them all at lookup time. IMAGE compound (arrayed) fields can be used as keys, and are grouped automatically.

For example, three fields containing phone numbers could be grouped together and would all be searched when retrieving by phone number.

Super-grouped retrieval

IMAGE can only search a single dataset at a time. SUPERDEX lets you form a super-group of a master set and one or more of its related (by IMAGE paths) detail sets and qualify master entries based on the contents of the related detail entries.

For example, a master set containing a book title related to a detail set containing authors and another detail set containing summary information could be super-grouped together, allowing master book entries to be qualified by title, author, and/or summary in a single operation.

Relational access: multiple criteria retrieval

Access may be performed using boolean operations against multiple criteria, to retrieve:

- all entries that meet either criterion (**OR** operation)
- all entries that meet both criteria (**AND** operation)
- all entries that meet one criterion but not the other (**AND NOT** operation)

For example, all the customers that have orders waiting to ship _or_ on back-order; all customers who are more than 60 days delinquent _and_ owe more than $1000; all parts that are out of stock _and not_ discontinued.

Relational access: multiple fields, sets, and bases

Relational queries may be performed based on multiple values across multiple fields, datasets, and databases using dynamically-joined indices. This provides the power of a relational database in accessing a regular IMAGE database.

For example, finding all the customers who have more than $100,000 in annual activity, current orders pending, and who did that same amount of business last year requires access to the CUSTOMERS and ORDERS sets in the SALES database and the ORDER-SUMMARY set in the HIST database.

Custom indexing

SUPERDEX contains a facility for addressing non-standard indexing requirements for circumstances in which the index value cannot be determined automatically.

Examples of this are data type conversion, date reformatting, upshifting, key extraction, and stripping unneeded characters.

Independent non-IMAGE indexing

SUPERDEX is designed to index entries in IMAGE databases, but can also be used to index other types of files.

For example, separate word processing documents may be indexed by all the significant words in their document descriptions and accessed via their file names.

Concepts

SUPERDEX looks and feels like IMAGE. It uses IMAGE-compatible intrinsics which have extended capabilities and, in some cases, additional *modes*.

These are the major concepts of SUPERDEX:

- **B-tree indices instead of chains**
  IMAGE uses doubly-linked lists to represent its chains. SUPERDEX uses *SUPERDEX indices* in B-trees which are contained in one or more standalone detail datasets in each database or in a separate database. These B-tree indices are automatically maintained and accessed by SUPERDEX intrinsics which are IMAGE-compatible. They are easy to configure and reconfigure.

- **B-tree = automatic master set**
  A SUPERDEX B-tree is functionally equivalent to an IMAGE automatic master set which provides access to a field in a dataset, commonly referred to as a "key". Like entries in an automatic master, SUPERDEX B-tree indices are added and deleted automatically. SUPERDEX easily replaces and enhances the functionality of automatic master sets with SUPERDEX indices.

- **Master and detail sets treated equally**
  In IMAGE, an automatic master may be related only to a detail set; in SUPERDEX, B-tree indices may be related to master sets as well as detail sets. In IMAGE, master and detail sets are handled differently: master sets are usually accessed via keyed reads (DBGET *mode* 7) and detail sets are accessed via DBFIND followed by DBGET *mode* 5 or 6. In SUPERDEX, both master and detail sets are accessed using a common method: DBFIND and DBGET *mode* 5 or 6, with DBFIND qualifying the entries and DBGET retrieving them--just like accessing an IMAGE path in a detail set. Of course, DBFIND and DBGET against IMAGE paths continue to function as in IMAGE.

- **Entries returned in sorted order**
  Entries are returned in ascending alphabetical order by SUPERDEX key value with DBGET *mode* 5 and descending order with DBGET *mode* 6.

- **Concatenated keys = sorted chains**
  A SUPERDEX key may consist of multiple field values or substring field values concatenated together, permitting more flexible sorting than sorted chains without the overhead.

- **SUPERDEX indices self-maintaining**
  SUPERDEX indices--like automatic master entries--are automatically added and deleted whenever DBPUT and DBDELETE are called. Additionally, DBUPDATE may also cause the indices to change automatically.

■ **Explicit index maintenance possible**
SUPERDEX indices may be added and deleted manually via new intrinsics. This permits custom indexing against IMAGE databases as well as indexing of external non-IMAGE files. This is called *Independent indexing.*

■ **Zero or multiple indices per entry**
A data entry may have zero or more SUPERDEX indices pointing to it, facilitating both multiple indexing (as used in keyword retrieval) and the exclusion of blank fields.

■ **Improved handling of compound items**
IMAGE does not allow compound items (which are also referred to as arrayed or repeating items) to be used as keys. In SUPERDEX, compound items may be used as keys and are handled such that every subitem in the item is automatically searched whenever the item is referenced.

■ **Power in the DBFIND mode 1 argument**
Most of SUPERDEX's powerful selection capabilities are available via DBFIND *mode* 1, with multiple values and operators included in the *argument* to define complex selection criteria. This permits generic code to be written and the user to specify the type and scope of retrieval.

■ **Selection refinement and undo**
SUPERDEX maintains the results of the current and previous DBFIND calls and manages them automatically; it also allows them to be manipulated explicitly. This permits successive DBFINDs to be used to refine and undo selections and to qualify entries across multiple fields, datasets, and databases.

■ **Multiple relational syntax used**
Boolean operations using multiple values can be specified in three common syntax.

SQL notation, as used with common SQL languages. The arguments can be entered with the words **AND, OR,** and **NOT.**

Infix notation, as used with common report-writers. The arguments can be entered with **+**, **−**, and **,** .

Reverse Polish Notation (RPN), as used by HP calculators. In RPN, the operator follows the two values to which it applies.

Terminology

Several new terms are used by SUPERDEX to identify its structures, and are used throughout this manual:

SI   SI stands for SUPERDEX index.

SI-key   Equivalent to an IMAGE search field, except in SUPERDEX the SI-key may consist of:

- a single field (*simple* SI-key)
- a substring field (e.g. only the first 6 characters of a 12-character field or the 3rd through 6th characters).
- a combination of up to four fields or substring fields (referred to as a *concatenated* SI-key), which permits extended sorting capabilities and may be searched by the entire concatenated key value or any portion thereof.

SI-subkey   A field or substring field used as an element in a concatenated SI-key. A simple SI-key, which references only one field, has no SI-subkeys.

SI-index   The B-tree entries which are comprised of the SI-key followed by an extension which points to the corresponding data entry.

SI-extension   Included at the end of the SI-index and used to map the corresponding data entries. For entries that reside in master sets, the SI-extension consists of the full IMAGE search field value, and its length is the same as the length of the search field. For detail sets, the SI-extension is the entry's relative record number, and is two words long.

SI-path   In IMAGE, a path defines the relationship between a master and detail dataset. In SUPERDEX, an *SI-path* defines any field (or combination of fields) that can be searched via SUPERDEX, as an IMAGE path would be used to index into a detail set. Entries along an SI-path are logically maintained in alphabetical order, so an SI-path may be thought of as a virtual sorted chain containing all the entries in the dataset.

SI-chain

In IMAGE, a chain is comprised of all the entries in a detail set that have the same search field value, as specified in the DBFIND *argument*. In SUPERDEX, an *SI-chain* is a virtual chain consisting of all the qualifying entries in a master or detail set that meet the search criteria as specified in SUPERDEX's DBFIND *argument* (which may or may not have the same IMAGE search field value).

SI-subset

Used only when performing Relational Access (boolean operations) against multiple values for a single SI-path, multiple SI-paths, datasets, and databases by performing successive DBFINDs. Both a virtual *active SI-subset* and *backup SI-subset* are maintained to contain the SI-chains retrieved by the DBFINDs.

SI-link

Used when performing Relational Access against multiple datasets; it defines the common item used to logically link the different sets.

The SI-link may also be used to enforce a sorting order when performing relational access against multiple SI-paths, sets, and bases.

It is required that the item assigned as the SI-link be configured an SI-subkey in a concatenated SI-key; alternately, for SI-paths against a master dataset, the SI-link may be the IMAGE search master field.

SI-counter

Optional parameter for the **I** *list* construct for DBGET which specifies how many SI-indices should be returned with a single DBGET call.

SI-definitions

Information about the SI-paths configured for a database.

SI-dataset(s)

One or more standalone detail datasets in each SUPERDEX'ed database which contain all the SUPERDEX B-tree structures. The *root* SI-dataset (named SI or SI0) contains the SI-definitions.

The SI-dataset(s) that contain the SI-indices may alternately reside in a separate database. For large databases or to optimize throughput, up to eight SI-datasets may be allocated, although one SI-dataset is normally sufficient. These datasets are named SI optionally followed by a sequence number (i.e. SI - SI7.).

SI-item

The only field in the SI-dataset(s), which is configured as a compound item named SI.

SI-index base

A separate database that contains the SI-indices, which may optionally be configured for any base. With this option, all SI-indices are maintained in the separate *SI-index base* rather than the primary base.

SI-pointer   A pointer in each B-tree that can be positioned before or after any index in the tree.

SI-intrinsics   The IMAGE-equivalent intrinsics used by SUPERDEX which are contained in an SL or XL (and provided in a USL) and referenced by all programs that use SUPERDEX.

Substring field   A partial definition of a IMAGE field. The starting character position and number of characters can be specified.

Access principles

Adding, updating, and deleting entries

In SUPERDEX, entries are added, updated, and deleted using its DBPUT, DBUPDATE, and DBDELETE intrinsics. These intrinsics are identical to their IMAGE counterparts and work exactly the same, except they also update any corresponding SI-indices.

Qualifying and retrieving entries

Entries are qualified and retrieved using SUPERDEX's DBFIND and DBGET intrinsics. These intrinsics are syntactically identical to the corresponding IMAGE intrinsics, but have extended capabilities and additional *modes*.

In SUPERDEX, DBFIND *mode* 1 may be called against a master or detail set with an *argument* that contains multiple values, and conditional and boolean operators. The qualifying number of entries is returned in the *status* array, and an internal SI-pointer is set in the B-tree. If the qualifying entry count is not needed, it is more efficient to instead use DBFIND *mode* 10. Additional DBFIND *modes* are available to perform specialized functions, such as setting a pointer to the alphabetical first or last entry in the set.

DBGET *mode* 5 may be used to retrieve the entries in ascending sorted sequential order; DBGET *mode* 6 in descending order. When all qualifying entries have been returned, an end-of-chain (or beginning-of-chain) condition is returned. New DBGET *modes* 15 and 16 may also be used to continue retrieving entries that are not on the SI-chain (those that no longer meet the search criteria).

Indexed access vs. relational access

Internally, one of two access methods is used in qualifying entries with DBFIND: *indexed access* or *relational access*.

Indexed access is used for retrievals that can be accomplished by accessing a single SI-chain. This accounts for most retrievals, and is used by default.

Relational access is used for boolean retrievals that require the use of multiple SI-chains, such as in performing retrievals against multiple SI-paths, sets, and bases by using multiple DBFIND calls.

Compatibility

Intrinsics

All SUPERDEX intrinsics are provided in an SL and XL (as well as a USL, RE-NUMRL, and NMOBJ files). All programs that access bases that are configured with SI-paths must reference these intrinsics, which are fully-compatible with their IMAGE counterparts.

Data types

SUPERDEX handles data as stored based on the IMAGE item data types. Search values may be represented in the same format as internally stored (ASCII, binary, etc.), or special conversion operators may be used.

☞ **Unsigned and signed values for items of data type P and Z are treated identically when qualifying entries.**

Status array

All SUPERDEX status information is returned in the standard IMAGE *status* array. The qualifying number of entries from DBFIND is returned in words 5-6 (like IMAGE), and end-of-chain and beginning-of-chain conditions are returned as condition words 15 and 16 in word 1 of the *status* array. *Status* words 7-10 are not available for SI-paths.

Error and exceptional condition handling

All errors and exceptional conditions are indicated by standard IMAGE error messages in the condition word field (word 1) of the *status* array.

Transaction logging

Because SUPERDEX uses IMAGE like intrinsics to maintain its SI-indices, it is covered by all forms of logging and recovery. Normal recovery methods recover not only the IMAGE chain pointers but also the SI-indices.

To further assure compatibility with transaction logging, for bases enabled for logging SUPERDEX automatically imposes DBBEGINs and DBENDs around all DBPUTs, DBUPDATEs, and DBDELETEs that do not already contain them.

Because SI-index maintenance generates additional log records, the SI-indices may optionally reside in a separate database (SI-index database) from the data entries they map . This results in no additional log records being written for the primary database, and logging can optionally be enabled or disabled for the separate SI-index database containing the SI-indices.

Application programs

Existing application programs require no changes or minor, straightforward modifications to utilize SUPERDEX's capabilities.

Because SUPERDEX is fully-compatible with IMAGE, all existing programs may reference SUPERDEX's SL or XL and continue to function as always. You can add SUPERDEX capabilities and introduce them to users over time, never having to run parallel with old applications or perform any significant conversion.

The simplest introduction of SUPERDEX is to replace all automatic master sets with SI-paths, which makes it possible to access the records in the related detail sets generically and in sorted order without any program modifications. The user need only include an @ and/or ? in the value being searched for. Another simple modification is to replace sorted IMAGE paths with concatenated SI-keys.

Third-generation languages

SUPERDEX supports programs written in COBOL, FORTRAN, Pascal, SPL, BASIC, Business Basic, and C. SUPERDEX will support programs written in RPG that call the IMAGE intrinsics directly.

Fourth-generation languages

SUPERDEX supports several 4GLs via special interfaces. TRANSACT and PROTOS are already compatible and only need to reference SUPERDEX's SL or XL. Front-end programs are available for Cognos' PowerHouse products QUIZ and QTP, while minor modifications to the dictionary facilitate SUPERDEX access via QUICK. Interfaces also exist for HP's Business Report Writer (BRW), Cogelog's VISIMAGE and Sydes' SYDAID. Interfaces to other 4GLs are currently being developed.

QUERY and other utilities

Most utilities that call IMAGE intrinsics, like QUERY/3000, are compatible with SUPERDEX in that any data added, deleted, or modified in these programs while referencing the SUPERDEX SL or XL will automatically adjust the associated SI-indices. The retrieval capabilities in utility programs, however, will vary based on various factors. The QUERY-like ASK2 utility from Cogelog is compatible with SUPERDEX.

Locking

Because the SI-indices may require change with every DBPUT, DBUPDATE, or DBDELETE, it is necessary to adopt an appropriate locking strategy whenever the SI-intrinsics are used.

SUPERDEX has several strategies for locking the SI-dataset(s) when necessary:

- you may add the SI-dataset(s) to the program's lock descriptor
- SUPERDEX will automatically lock and unlock the dataset containing the data and the SI-dataset in succession
- SUPERDEX will implicitly lock the SI-dataset when necessary (requires MR capability)
- SUPERDEX may be configured to perform a separate DBOPEN for each process accessing a database and perform its locking via that "access path" (requires MR capability)
- SUPERDEX may be configured to maintain the SI-indices in a separate base, in which case a separate DBOPEN is performed against the primary base and the SI-index base (requires MR capability)

Refer to the discussion in the _Locking_ chapter of the _Programming_ section for a complete description of the various locking strategies and their advantages and disadvantages.

Capabilities

SUPERDEX uses standard IMAGE intrinsics for maintaining its SI-indices, which means it does not require Privileged Mode and will be compatible with future releases of IMAGE and MPE.

All programs that reference the SUPERDEX SL or XL--except Native Mode programs under MPE/XL, which do not require DS capability--must have DS and conditionally MR capability, depending on the locking method used. The groups and accounts in which these programs reside also require these capabilities, as do users who :PREP programs for use with SUPERDEX.

Stack requirements

SUPERDEX requires about two Kwords of stack space for indexed access and one or two additional Kwords for relational access (if used). Programs that are short of stack space may require MAXDATA increases. Programs that already utilize the maximum MAXDATA may need to be :RUN with the ;NOCB parameter.

Native Language support

SUPERDEX fully supports Hewlett-Packard's Native Language Support facility for matching, collating, and other significant operations.

7-bit support for Swedish language

☎ A special version of SUPERDEX has been developed for support of the Swedish language. Since new files must be supplied, please contact Bradmark if you need this version.

Section 2	**Access methods**

Overview

This section looks at various methods available in SUPERDEX for accessing entries in IMAGE databases.

Each chapter covers a different access method and discusses the functionality provided, its applications, details about configuration and implementation, rules of operation, efficiency, and maintenance considerations. Examples are given throughout.

Chapter 1   Multiple keys in master and detail datasets
Function   For accessing data entries in manual master and detail datasets by any number of keys.

Chapter 2   Concatenated SI-keys containing multiple fields
Function   Up to four fields or substring fields may be concatenated together to form a composite key.

Chapter 3   Sorted sequential retrieval
Function   All entries are returned in ascending alphabetical or descending alphabetical order.

Chapter 4   Keyword retrieval
Function   Permits an entry to be accessed by any significant word contained in any keyworded key.

Chapter 5   Generic and partial key retrieval
Function   Allows entries to be searched for by the first few letters of a key or by a string embedded in a key.

Chapter 6   Approximate match retrieval
Function   SUPERDEX can find the nearest matching entry if no entry that matches a specified value exists.

Chapter 7   Greater-than, less-than, and range retrieval
Function   Permits searches for all entries that are greater than or equal to, less than or equal to, or not equal to a specified value, or that fall within the range of two values.

Chapter 8   Grouped retrieval
Function   For handling multiple fields that are functionally equivalent as one logical field at lookup.

Chapter 9   Super-grouped retrieval
Function   For allowing master entries to be qualified based on their contents and the contents of their related detail dataset entries.

Chapter 10 Relational access: multiple criteria retrieval

Function   Permits multiple values to be specified for a field in a single lookup operation and the results to be combined by using boolean operations.

Chapter 11 Relational access: multiple fields, sets, and bases

Function   Like multiple criteria retrieval, permits multiple values and boolean operators to be specified--but extends these capabilities to work on multiple fields, datasets, and databases.

Chapter 12 Custom indexing

Function   Indices may be calculated by a user-written procedure, providing complete flexibility in indexing entries.

Chapter 13 Independent indexing

Function   Permits non-IMAGE data to be indexed by SUPERDEX, for purposes such as document management.

Multiple keys in master and detail datasets

Functionality

IMAGE lets you access a master set by only one field and a detail set only via its related masters unless time-consuming serial reads are used, forcing rigid applications and cumbersome database structures. SUPERDEX lets any dataset be accessed quickly by any field or fields--even every field.

Application

Due to the inability to specify more than one field in accessing a master dataset and to the high overhead of detail dataset paths, access to datasets is typically restricted to either one or a few fields.

For example, customers stored in a master dataset may be accessed only by the customer number (unless a serial read is performed). In SUPERDEX, they could be accessed by various fields, such as COMPANY-NAME, CONTACT-1, CONTACT-2, and PHONE. If the customers were instead contained in a detail dataset with several related automatic master datasets, they could stay there, and the related automatic master sets could be replaced by SI-paths with the same names as the search fields and accessed via DBFIND *mode* 1, meaning that no program modifications would be required to facilitate partial-key access and other powerful retrievals.

Implementation

Each field or combination of fields that is to be accessible as a key must be configured in an SI-path, and SUPERDEX creates a B-tree for each one.

For simple, single-field SI-keys, the entire field could form the index or a substring of the field could be used.

Operation

DBPUT, DBUPDATE, and DBDELETE automatically maintain the SI-indices in the B-trees.

Entries are qualified by DBFIND *mode* 1 or one of several new *modes* and retrieved by DBGET *modes* 5 and 6 or new *modes* 15 and 16. If the *item* parameter contains the name of an SI-path, the B-trees are automatically accessed; otherwise, regular IMAGE access is performed.

Efficiency

It is recommended that SI-keys be kept as short as possible for efficiency. The longer the SI-key, the more access will be necessary to manage the associated SI-indices. Substring SI-keys should be used where possible. For example, 10 characters of a 20-character LAST-NAME field may be sufficient for indexing purposes, or only the 5th character through the 9th character of a general ledger account number may be sufficient.

Concatenated keys containing multiple fields

Functionality

IMAGE restricts a key to a single field. SUPERDEX permits multiple fields or substring fields to be concatenated together and retrieval to be done on the entire concatenated value or any portion thereof.

Up to four fields, or substring fields, may be concatenated to form an SI-key, permitting enhanced retrieval and sorting capabilities and eliminating the need for sorted IMAGE paths and many programmatic sorts. More than four fields may be included in a concatenated key as long as they are physically contiguous in the dataset.

Application

Concatenated SI-keys permit very specific lookups to be performed without having to read down a chain to qualify entries that match on multiple fields because all fields may be contained as SI-subkeys in the SI-key.

For example, a customer number and order date could be concatenated together to form an SI-key. To access the SI-key, a single composite value would be specified and the corresponding entry returned. The alternative in IMAGE would be having to read down the customer's chain until the order with the specified date was encountered.

Implementation

The combination of fields that are to be accessible as a concatenated SI-key must be configured in an SI-path, and SUPERDEX creates a single B-tree for the concatenated SI-index.

☞ **If related to a detail set, a concatenated SI-key may consist of up to four fields or substring fields. For master sets, up to three fields (four if the IMAGE search field is included) may be defined. Each field or substring field is called an *SI-subkey*.**

If more than four fields are needed for a concatenated key, SUPERDEX is able to support this, so long as the extra fields are contiguous in the dataset. This is facilitated when configuring the SI-path by declaring an SI-subkey length that exceeds the length of the specified field. This results in the specified number of characters being included in the SI-subkey, thereby forming an SI-subkey that contains multiple fields or truncated fields. To utilize this feature, it is necessary to set a special JCW named SIEXTLEN during SIMAINT operation.

Operation

DBPUT, DBUPDATE, and DBDELETE automatically maintain the SI-indices for concatenated SI-keys.

Entries are qualified by DBFIND *mode* 1 or one of several new *modes* and retrieved by DBGET *modes* 5 and 6 or new *modes* 15 and 16. If the *item* parameter contains the name of an SI-path, the B-trees are automatically accessed; otherwise, regular IMAGE access is performed. The entire combined SI-key value or a partial value may be specified in the *argument* parameter.

If the concatenated SI-key has been configured using the **SIEXTLEN** JCW to include more than four SI-subkeys, it is required that all fields that are included in the concatenated SI-key but are not explicitly referenced by name be included in the DBGET *list* in the order in which they occur in the dataset, for preparation for DBUPDATE and DBDELETE.

Efficiency

It is recommended that each SI-subkey in an SI-key be kept as short as possible for efficiency. Substring SI-subkeys should be used where possible. Concatenated SI-keys are generally less efficient than simple SI-keys because they typically have longer lengths and therefore cannot be managed as efficiently; however in many instances they can outperform simple SI-keys because they can significantly reduce the number of entries qualified.

Sorted sequential retrieval

Functionality

IMAGE returns detail chain entries in chronological order, unless sorted paths are used.

SUPERDEX returns entries in ascending or descending alphabetical order, providing a natural sorting mechanism. The sort criteria may be further extended by using concatenated SI-keys, which provide more flexibility than sorted paths without the overhead.

Application

The ability to retrieve entries in sorted sequential order eliminates the need for many or all program sorts, and requires no special handling.

The alphabetic ordering of SI-indices also permits approximate match retrieval (described later).

Operation

Entries are unconditionally returned in sorted sequential order for entries qualified in indexed access mode.

In relational access mode, an SI-link may be specified in the *item* parameter of DBFIND to enforce a sorting order.

Entries are returned in ascending sorted sequential order with DBGET *modes* 5 and 15 and descending sorted sequential order with *modes* 6 and 16. All the entries in a dataset may be read in ascending or descending sorted order by calling DBFIND *mode* 100 or 200, respectively, and DBGET *modes* 15 or 16.

SUPERDEX uses HP's Native Language Support facility in returning data contained in alphanumeric (data types X and U) items for databases in which NLS is enabled, assuring that language-specific attributes (such as esstsets and umlauts) are handled properly. The language is determined from the root file of each database, and may be established in the database schema or by DBUTIL. The collating sequences used by SUPERDEX, including language-dependent variations, are documented in HP's **Native Language Support Reference Manual**.

☞ **SUPERDEX respects the sign in sorting data contained in numeric items (data types I, J, P, R, and Z), and thereby returns negative values before positive values.**

Keyword retrieval

☞ **Keyword retrieval is available only in the SUPERDEX II package.**

Functionality

SUPERDEX lets you access SI-keys by any significant word they contain for SI-paths configured as keyworded. For example, an entry with the SI-key value "REDUCED INSTRUCTION SET COMPUTER" could be located by the values **REDUCED, INSTRUCTION, SET,** or **COMPUTER.**

Application

Keywording is useful for indexing fields that contain multiple values, such as company names, street addresses, last/first names, part descriptions, and comments.

Implementation

☞ **Keywording can be implemented on any** *alphanumeric* **(data type U or X) item, for a simple or concatenated SI-key. For a concatenated SI-key, only the first SI-subkey is keyworded.**

Keyworded SI-paths are configured in the SIMAINT program by appending **/K** to the SI-path name. You must specify the *keyword length*, which refers to the maximum number of **characters** to include in the SI-key. For indexing purposes, words that are longer than the keyword length are truncated; those that are shorter are padded with spaces.

Also specified is the *minimum number of characters per keyword*, which defines the minimum number of characters, between 1 and 4, that a word must contain in order to qualify for keywording. This permits very short words to be easily excluded based on their length.

Additionally for each keyworded SI-path, the *average number of keywords* must be specified, which refers to the average number of significant, unique keywords within each SI-key, between 1 and 16. If requirements change at a later time, the average number of keywords may be changed by reorganizing the SI-path--a new value may be entered at that time.

To eliminate unnecessary or common words by value from keywording, an exclusion list can be defined which restricts the keyword entries to only relevant ones. Exclusion words are specified in a disk file via any editor and uploaded into a special standalone SI-path called **KWEXCLUDE.**

Operation

For SI-paths that are defined as keyworded, every word in the SI-key separated by spaces or special characters is treated as a keyword. **(You may optionally specify up to four special characters to be excluded as keyword delimiters when configuring SI-paths.)** Multiple SI-index entries (one for each unique value in the SI-key) are automatically generated by DBPUT and removed by DBDELETE. For compound IMAGE items that are keyworded, each subitem is examined separately and keyworded accordingly. **For concatenated SI-keys, only the first SI-subkey is keyworded.**

All keywords are upshifted for indexing and matching purposes.

Keyworded fields are always searched by individual significant word during the DBFIND operation. Additionally:

- words that contain a hyphen are keyworded multiple times: once for each hyphen plus one. For example, "HEWLETT-PACKARD" would be multiply indexed and could be located by both **HEWLETT-PACKARD** and **PACKARD**, and "TIC-TAC-TOE" could be located by **TIC-TAC, TAC-TOE**, and **TOE**. (This multiple-indexing feature can optionally be disabled when configuring SI-paths.)

- SI-keys in which the same word appears more than once are indexed only once for that word

- a maximum of 16 keywords per SI-key (for simple SI-keys) or SI-subkey (for concatenated SI-keys) is allowed.

Keyworded SI-paths are accessed in the same way as non-keyworded SI-paths--the only difference is in the configuration of the SI-path.

Efficiency

Keyword lengths should be kept as short as possible, typically 5 or 6 words, for efficiency. The minimum keyword length should be set at 4, if possible, to exclude very short words that contain less than four characters.

Commonly occurring special characters should be excluded as keyword delimiters to avoid unnecessary indexing. For example, if keywording entries in which dates are common (e.g. "02/20/90"), the slash character (/) should be excluded.

If multiple indexing of hyphenated values is not required in order to locate entries, this feature should be disabled. This is especially significant for SI-keys in which hyphens are very prevalent, such as part numbers (e.g. "123-999-447").

Also, only very common words should be configured for exclusion, since the required overhead when entries are added increases with the number of excluded words.

Maintenance

All keyworded SI-paths and the **KWEXCLUDE** (exclusion word) SI-path must be reorganized whenever any changes are made to the file of excluded words.

☞ **The keyword exclusion file must be present in the same group/account as the database when SI-paths are reorganized.**

Generic and partial key retrieval

Functionality

IMAGE will not find an entry unless you specify its key value exactly in its entirety. SUPERDEX permits a partial key or keyword to be specified, as well as a generic key containing wildcards.

Application

Probably the most requested capability for IMAGE databases is generic and partial key access: the ability to specify only a few significant characters of the key rather than its entire value.

This saves not only time and keystrokes, but locates entries whose exact values are not known or which cannot be located due to misspellings or other reasons.

Generic key access permits values that match a specified pattern to be located, useful for selecting entries with commonalty. Partial key access allows for a variable number of positions to be defined.

Implementation

Generic and partial key retrievals may be performed on any *alphanumeric* field (data type X or U) referenced in an SI-path. They may not be performed against numeric fields (data types I, J, K, P, R and Z).

Operation

Partial key access can be performed by three different methods:

The first is to specify the partial key value appended with an @ as the *argument* for DBFIND *mode* 1, e.g. **HEWL@**. DBFIND will locate all entries that match on the significant characters followed by anything. (A character other than @ may be designated as the wildcard character when configuring SI-paths.)

The second method is to specify up to two (2) @ in the *argument*, surrounded by << >> for DBFIND *mode* 1, e.g. **<<H@L@T>>**. DBFIND will qualify all entries that contain all three groups of significant characters in the specified order.

The last method is to specify the value in the *argument* without an @ but vary the *mode* based on the length of the *argument*. For example, an *argument* containing the partial key **ROLA** would dictate *mode* 102 or -104 (100 plus the number of words or bytes, respectively, in the value).

Generic key retrieval is accomplished by embedding the **?** or **#** matchcodes in the *argument*.

The **?** holds the place of any alphanumeric character. For example, the *argument* **L?TTER** would locate "LETTER" and "LITTER"; by appending an **@** (**L?TTER@**), "LETTERMAN," "LITTERBUG," and "LOTTERY" would also be located. (A character other than **?** may be designated as the matchcode when configuring SI-paths, or the single-character matchcode may be disabled.)

The **#** holds the place of only numeric characters. The *argument* **AP#J@** would locate "AP2J8A99" and "AP7JIT", but not "APYJ97K".

Efficiency

Search *arguments* that contain one or more **?**s or **#**s in the leftmost character positions or contain an **@** in the first character position are less efficient than those that begin with alphanumeric characters. Therefore, for best performance, a substring field should be specified.

Approximate match retrieval

Functionality

Neither IMAGE nor SUPERDEX can find an entry that does not exist, but SUPERDEX can do the next best thing: find the nearest matching entry.

Application

Approximate match retrieval, like partial and generic key retrieval, is useful in circumstances in which the exact key value is not known. Unlike partial and generic key retrieval, approximate match retrieval does not require that any entry matching the specified value exist: the nearest matching entry is always found.

For example, if the value **UNITED** is input and no matching entry exists, the nearest matching entry in ascending or descending order, "UNIFIED" or "UNITY," may be retrieved.

Implementation

The sorted ordering of SI-indices permits approximate match retrieval by using new SUPERDEX DBFIND *modes*. If no entry that matches the search criteria exists, the internal SI-pointer is set in the B-tree to the nearest qualifying entry, permitting a program to start reading entries at any alphabetic location in either ascending or descending order.

Approximate match retrieval may be performed on any *alphanumeric* item (data type X or U) referenced in an SI-path.

Operation

Approximate match retrieval is performed by using a DBFIND *mode* that specifies how many characters in the *argument* SUPERDEX should match on, which is typically the length of the value specified. If no matching entry exists, the nearest matching entry is returned.

The *mode* also dictates whether the internal SI-pointer in the B-tree should be set before or after the matching or nearest matching entry, permitting subsequent DBGETs to include or exclude that entry.

For example, an *argument* containing the value **UNITED** would dictate *mode* 103 or -106, both of which would cause DBFIND to match on the entire value. The *mode* is calculated as 100 plus the number of words or bytes (negated if bytes). Using these *modes*, the SI-pointer would be set <u>before</u> the matching or nearest matching entry. Subsequent DBGETs in ascending order (*mode* 15) would include any entries beginning with "UNITED," while DBGETs in descending order (*mode* 16) would exclude them.

With a *mode* of 203 or -206 (200 plus the number of words or bytes), the SI-pointer would be set <u>after</u> the matching or nearest matching entry, and the "UNITED" entries would now be included with subsequent DBGETs in *mode* 16 (descending) but would excluded be included with *mode* 15 DBGETs (ascending).

Greater-than, less-than, and range retrieval

Functionality

In addition to generic and partial-key retrieval, SUPERDEX permits retrievals of entries that are

- greater than or equal to a specified value
- less than or equal to a specified value
- not equal to a specified value
- in the range between two values

Application

Greater-than-or-equal-to and less-than-or-equal-to retrievals are especially useful for operations against amounts, such as finding all customers with balances of $1000 or more.

Not-equal-to retrieval is useful for testing for the absence of a value for a particular field, such as all invoices that are not "PAID."

Range retrievals may be used against ordered values, and can be used, for example, to find all customers in a given geographical area by means of a range of zip codes. In addition, pattern matching is supported within a range retrieval which is useful, for example, for finding all orders for a given customer within a date range where the SI-key is a concatenation of the date and customer number.

Implementation

Greater-than-or-equal-to, less-than-or-equal-to, not-equal-to, and range retrievals may be performed against both alphanumeric and numeric items.

☞ **They operate on any value in any SI-key, including keyworded SI-keys.**

Operation

These retrievals are performed by embedding special operators in the *argument* for DBFIND *mode* 1 or 10.

Greater-than-or-equal-to retrieval is accomplished by prefixing the argument with the >= operator (e.g. **>=1000**), less-than-or-equal-to retrieval uses the <= operator as a prefix, and not-equal-to retrieval uses the <> operator as a prefix. The <> operator can also appear after another value in the same *argument* to exclude records (e.g. **SUPER@<>SUPERDEX**).

Range retrievals are performed by using the >= and <= operators in combination. For example, a range search to find all the entries with amounts between 500 and 1000, inclusive, is specified with the *argument* >=500<=1000. Pattern matching may be done within a range by specifying the pattern, start point, and endpoint in the *argument*; for example, an *argument* of ??????4433>=890101<=891231 against a concatenated key containing date and customer number would find all the orders for the customer 4433 placed in 1989.

Entries may be retrieved in ascending or descending sorted order with DBGET *modes* 5 and 6, which return end-of-chain and beginning-of-chain conditions when all entries have been read.

Greater-than-or-equal-to and less-than-or-equal-to retrievals may alternately be accomplished without specifying the >= and <= operators and instead using any DBFIND *mode* and *argument* followed by DBGETs with new modes 15 and 16, which perform greater-than ascending and less-than descending retrievals, respectively.

Grouped retrieval

☞ **Grouped retrieval is available only in the SUPERDEX II package.**

Functionality

IMAGE can search only one field at a time. SUPERDEX lets you group multiple fields in a dataset together at configuration time, and automatically searches them all at lookup time, thereby handling them as one logical field. By default, SI-keys are not grouped.

This grouping technique is automatically imposed on all compound IMAGE items used in SI-keys, and generates a separate SI-index for each subitem value. The result is that every subitem is always searched automatically whenever the item is referenced.

Application

Grouping is useful for logically combining multiple fields in a dataset that are functionally identical.

For example, a two-line address may be stored in the fields ADDRESS-1 and ADDRESS-2 with addresses contained on either or both lines. The two fields may be configured as SI-keys and grouped together in an SI-path called *ADDRESS*, and both will be searched automatically whenever *ADDRESS* is referenced.

Or, if a quick customer lookup mechanism is needed in which either the company name, contact name, or phone number may be specified in response to a single prompt, the fields COMPANY, CONTACT, and PHONE could be grouped together under the SI-path name *QUICK-LOOKUP*.

Implementation

Multiple SI-keys may be grouped together into a single SI-path as a configuration option in SIMAINT by appending **/G** to the SI-path name.

☞ **All SI-keys contained in a group must be of the same data type. It is also required that each SI-key in the group be configured with the same length, so it is possible that some SI-keys must be substringed and others padded with spaces.** For example, if COMPANY is X30, CONTACT is X20, and PHONE is X14, these three fields may be grouped together with any length between 7 and 15 words--with 7 words, the SI-key for both COMPANY and CONTACT would be truncated; with 15 words, both CONTACT and PHONE would be padded with spaces; or any length in between could be chosen.

In configuring a group, specify the longest SI-key first. It will establish the length of the grouped SI-path, as its length (or substring length, if specified) is unconditionally applied to subsequently configured SI-keys belonging to the same group.

☞ **SI-keys that are used as IMAGE master set search fields should be specified last.**

Concatenated SI-keys can also be grouped. The second through the fourth SI-subkeys are repeated for each SI-key in the group. This allows COMPANY and CONTACT to be grouped and to have the LAST-ACTIVITY-DATE concatenated with both indexes.

Operation

Whenever the group is referenced by its SI-path name in the *item* parameter of DBFIND, all SI-keys that form the group are unconditionally searched.

Grouped SI-paths are accessed in the same way as non-grouped SI-paths--the only difference is in the configuration of the SI-path.

There may be some ambiguity in searching by an SI-key in a grouped SI-path whose item length is shorter than the group length and which is therefore padded with spaces. For example, if CITY, an X16, and STATE, an X2, are grouped together with an SI-key length of 8 words (to accommodate CITY), an *argument* of **CA** would find not only all entries in the state of "CAlifornia" but also those in the cities of "CALABASAS" and "CARLSBAD." To resolve this ambiguity, use DBFIND *mode* 1 or 10 and pad the *argument* with enough trailing spaces to cover the full SI-key length.

Super-grouped retrieval

☞ Super-grouped retrieval is available only in the SUPERDEX II package.

Functionality

IMAGE can search only a single field in a single dataset at a time. SUPERDEX lets you group together a master dataset with one or more of its related detail sets at configuration time, and automatically searches the configured fields in the related detail datasets at lookup time.

The result is that entries in master datasets can be qualified based on the values in related detail sets.

Application

Super-grouping is useful for qualifying master entries based on a logical combination of each master entry and its related detail entries, or on just the related detail entries.

For example, a library system may contain the title of a book in a master dataset, a description of the book on multiple entries in a related detail set, and the book's author(s) on one or more entries in another related detail set. The master set is keyed on book number and pathed to the two related detail sets on the same item. Together, the master and detail entries of a given book number form a profile of the book.

A super-grouped SI-path called *BOOK-PROFILE* could be configured based on the book name (from the master dataset), description (from one detail dataset), and authors (from the other detail dataset), permitting a book to be qualified by one or more of its characteristics (such as author plus title). The greatest functionality would be achieved by defining the super-grouped SI-path as keyworded, permitting any word in the book name, summary, or author--or any combination thereof--to be specified for retrieval.

It is not required that the master entry itself be contained in the super-group: it is possible to define a super-group in which the detail entries only are used to qualify their related master entries. This means that a DBFIND against the master dataset can only be qualified by values in the detail dataset.

The super-group in the previous example could have alternately been defined to consist of only the descriptions and authors (excluding the book name). This way, books could be qualified by author and/or description.

Implementation

Multiple SI-keys in related datasets may be grouped together into a single SI-path as a configuration option in SIMAINT by first defining the SI-path for the master dataset and then referencing this SI-path (by appending /G to the SI-path name) for each detail dataset that should be included in the super-group.

If instead defining a super-group that does not include the master entry itself, the SI-path is specified for the master set using an item from one of the related detail sets (since no item from the master set is included in the super-group).

☞ **All detail sets included in the super-group must be related to the master dataset by an IMAGE path. Also, the name of the item in each detail dataset that forms the path must be the same as the name of the search field in the master dataset.**

Concatenated SI-keys can also be super-grouped. The second through the fourth SI-subkeys are repeated for each SI-key in the super-group.

Operation

DBPUT, DBUPDATE, and DBDELETE to all datasets configured in the super-group automatically maintain the SI-indices.

Super-groups may be accessed only via the master dataset--not the related detail sets. So DBFIND must be called against the master set, referencing the name of the super-group in the *item* parameter. DBFIND automatically searches all SI-keys in all (master and related detail) datasets in the super-group and qualifies the corresponding master entries based on them.

Only qualifying master entries are returned by SUPERDEX's DBGET--not their related detail entries. If desired, use standard IMAGE DBGETs (mode 5 or 6) to read the IMAGE chains to retrieve the detail entries related to the qualifying masters.

As illustrated, **SUPERDEX's DBFIND and DBGET qualify and retrieve only master entries.** The detail entries in the super-group are only used as criteria for qualifying their related master entries.

Maintenance

Whenever the SIMAINT program is used to reorganize or delete any SI-key contained in the group, all SI-keys in the group are automatically reorganized or deleted.

Relational access: multiple criteria retrieval

☞ **Relational access retrieval capability using multiple criteria is available only in the SUPERDEX II package.**

Functionality

SUPERDEX can search an SI-path for a combination of multiple search criteria in a single operation.

Retrievals against multiple criteria can be used to locate entries:

- that meet either criterion (Boolean **OR** operation)
- that meet both criteria (Boolean **AND** operation)
- that meet one criterion but not the other (Boolean **AND NOT** operation)

Application

Often, it is not enough to be able to specify keys in partial or generic format; rather, it is necessary to locate entries that meet multiple criteria.

Boolean operations provide the most powerful and flexible search capability. Some examples would be finding all entries in a keyworded SI-path that contain both of two keywords (AND operation), or one keyword or the other (OR operation), or one keyword and not the other (AND NOT operation).

Implementation

Boolean operations may be specified against any SI-path, regardless of its configuration.

Operation

Boolean operations are accomplished by embedding the appropriate notation in the DBFIND *argument*.

For example, a search for all part descriptions that contain both the words PAPER and CLIP could be specified with an *argument* of either `˜PAPER AND CLIP;` (SQL Notation), `˜PAPER+CLIP;` (Infix Notation), or `[PAPER][CLIP]&` (Reverse Polish Notation). To find all invoices that are unpaid or cancelled, the *argument* would be `˜UNPD OR CANC;`, `˜UNPD,CANC;`, or `[UNPD][CANC]|`. Additionally, to find all entries in California and not Los Angeles would be specified as `˜CA NOT "LOS ANGELES";`, `˜CA-"LOS ANGELES";`, or `[CA][LOS ANGELES]!&`.

Values stored in binary may be qualified by specifying the search values in ASCII format.

For a more complete description of boolean operations, refer to the _Qualifying entries with DBFIND_ chapter in the _Programming_ section of this manual.

Efficiency

When performing boolean operations using multiple values, it is always recommended for efficiency to specify the less common value or values first. For example, an _argument_ of **˜JOHN AND BROWN;** causes SUPERDEX to select all the entries that contain "JOHN" and then deselect those that do not contain "BROWN". If there are more records with JOHN, it would be considerably faster and more efficient to specify **˜BROWN AND JOHN;** instead, since far fewer entries would be selected in the first lookup.

Relational access: multiple fields, sets, and bases

☞ **Relational access retrieval capability using multiple fields, sets, and bases is available only in the SUPERDEX II package.**

Functionality

Relational queries may be performed using multiple values across multiple fields, datasets, and databases using dynamically-joined indices. This provides the power of a relational database within a regular IMAGE database.

Application

These features permit entries to be located by multiple criteria on multiple fields in a dataset, as well as using multiple datasets and databases to qualify entries.

For example, to find all unpaid invoices in the ORDERS database with amounts greater than $1000 might require testing both the PAID-FLAG and ORDER-TOTAL-AMT fields of the INVOICE-HEADER dataset.

To find all those invoices only for customers with poor payment history would also require a lookup in the AVG-DAYS-TO-PAY field in the CUSTOMER dataset. And to attain more complete information about these customers may require access to entries archived in the ORHIST database.

Implementation

Entries are qualified based on multiple fields, sets, or bases with multiple DBFIND calls against any SI-paths. No special SI-path configuration is required.

Operation

Multiple DBFIND calls are performed in succession, with one DBFIND per SI-path with varied *base*, *dset*, and *item* parameters which specify the database, dataset, and SI-path to access.

In the preceding example, four DBFINDs would be performed in succession: one against the *PAID-FLAG* SI-path in the ORDER-HEADER set in the ORDERS base, another against the *ORDER-TOTAL-AMT* SI-path, another against the *AVG-DAYS-TO-PAY* SI-path in the CUSTOMERS set, and the last against the dataset and SI-path of the same name but in the ORHIST database.

Custom indexing

Functionality

SUPERDEX indexes each entry based on its configured SI-paths using the literal value of each SI-key, as influenced by truncated fields, concatenated SI-keys, and keyworded SI-paths.

There are circumstances in which this may not be sufficient to properly index an entry and where additional intelligence is required to compose the SI-key.

To address these requirements, SUPERDEX allows entries to be indexed by any value that may be calculated from the data entry.

Application

Some examples of requirements for customized SI-indices are:

- data type conversion
- reformatting date (e.g. ASCII to Julian)
- upshifting
- specialized SI-key extraction (embedded key)
- stripping unneeded characters
- facilitating concatenated SI-keys comprised of more than four SI-subkeys

Implementation

SUPERDEX provides an exit in the form of a user-written procedure that permits SI-indices to be calculated by parsing any values in data entries.

This procedure, named **SIUSER**, needs to be written by the user and placed in the SL or XL along with the SUPERDEX intrinsics.

Operation

The SIUSER procedure is called unconditionally with every DBPUT, DBUPDATE, DBDELETE, and by the SIMAINT utility, and thereby automatically maintains the generated SI-indices.

Custom SI-keys should be handled as ASCII, since sorting is done using the binary or NLS representation and non-ASCII SI-keys would not sort properly. For binary SI-keys, the DBFIND *argument* must also be specified in binary format and *modes* 1*nn* or 2*nn*, or *modes* 1 or 10 with the full SI-key value, may be used.

Independent indexing

Functionality

Independent indexing describes the use of SUPERDEX to index entries contained in a structure other than an IMAGE database, such as flat MPE files. This permits SUPERDEX's advanced data qualification capabilities to be used on data external to an IMAGE database.

Application

SUPERDEX's independent indexing facility is intended to index files that supplement an IMAGE database, although an IMAGE database need only exist to house the SI-dataset in which the SI-indices are maintained.

For example, a document management system that manages separate word processing files could be implemented using independent indexing, with the SI-index consisting of the article title as the SI-key and the file name as the SI-extension. This permits an article to be looked up by its title, and its file name returned to the calling program.

Implementation

Independent indexing is implemented by configuring a standalone B-tree for each independent SI-path using the SIMAINT program. The dataset is left blank, and the SI-path name and SI-index length, including the SI-extension, are defined.

☞ **With independent indexing, the type and value of the SI-extension is unknown to SUPERDEX and must be specified, since the file and data structures are designed by the user. Typically, a file name or record number is used as the SI-extension.**

Operation

The entities being indexed are added and deleted by some method that is unknown and of no concern to SUPERDEX.

SI-indices must be explicitly added and deleted using the new DBPUTIX and DBDELIX intrinsics, for which both the SI-key and SI-extension are specified in the *buffer* parameter. The database that contains the SI-indices is specified in the *base* parameter, and the SI-path in the *item* parameter. The *dset* parameter is left blank or set to **200**.

Entries are qualified and retrieved by DBFIND and DBGET. For both, the *base* parameter specifies the database that contains the SI-indices, the *dset* parameter is left blank or set to **200**, and the SI-path is defined in the *item* parameter. For DBGET, the **!** *list* is used to return the entire SI-index, including the SI-extension.

Independent SI-indices should be handled as ASCII, since sorting is done using the binary or NLS representation and non-ASCII SI-indices would not sort properly. For binary SI-indices, the DBFIND *argument* must also be specified in binary format and *modes* 1*nn* or 2*nn*, or *modes* 1 or 10 with the full SI-index value, may be used.

Maintenance

Correspondence of the SI-indices to the entities they reference is the sole responsibility of application programs. No method is provided for implicitly manipulating the SI-indices, nor maintaining their synchronization.

Section 3 Configuration / Establishing SI-indices

Overview

This section describes the methods used in configuring SUPERDEX for your databases. This includes establishing the SI-item and SI-dataset(s), defining SI-paths, and establishing B-trees and SI-indices.

This section assumes that you have already loaded SUPERDEX on your system, as described in the separate *SUPERDEX loading instructions*.

Chapter 1 Installation
Function describes procedures, including SL creation and modifications to users, groups, accounts, programs, job streams, menus, and UDCs.

Chapter 2 Configuration overview
Function provides a brief description of the various configuration options, including simple vs. concatenated SI-keys, handling of compound IMAGE items, SI-path names, SI-key lengths, and various restrictions.

Chapter 3 Excluding words from keywording
Function reviews the process of excluding words from keywording and shows how to create the KWEXCLUD file and configure exclusion words.

Chapter 4 Customizing default characters
Function illustrates customizing default characters to optionally redefine the characters used to represent the wildcard and matchcode, to disable multiple indexing of hyphenated values for keyworded SI-paths, and to exclude certain special characters from being recognized as keyword delimiters.

Chapter 5 Configuring SUPERDEX using SIMAINT
Function describes the methods of configuring SUPERDEX using SIMAINT and discusses program operation, access requirements, and input rules. It then gives examples of how to define various simple and concatenated SI-paths for keywording, grouping, and other functions, as well as custom and independent SI-paths.

Options for maintaining SI-paths are not discussed in this section--refer to the *Maintenance and utilities* section for information.

Installation

Once the SUPERDEX installation tape has been loaded, as described in the separate _SUPERDEX loading instructions_, several steps are necessary for completing the SUPERDEX installation.

Some of these steps are one-time operations, while others are required for future creation of users, accounts, programs, etc. The installation procedures may include:

- creating SUPERDEX SL(s) or XL(s)
- creating a group for utility programs
- adding capabilities to accounts and groups
- adding capabilities to users
- adding capabilities and stack space to programs
- changing menus, UDCs, and job streams
- expanding system tables

Creating SUPERDEX SL or XL

The SUPERDEX SL is SL.PUB.SUPERDEX; the XL (for Spectrum MPE/XL systems) is XL.PUB.SUPERDEX. Copy the SL or XL into the PUB group of every account that contains programs which will use SUPERDEX.

If SL.PUB._youracct_ already exists, add the SUPERDEX segments into it from the file SIUSL.PUB.SUPERDEX:

```
:SEGMENTER
-SL  yoursl
-USL SIUSL.PUB.SUPERDEX
-ADDSL SIPROC0
-ADDSL SIPROC1
-ADDSL SIPROC2
-ADDSL SIPROC3
-ADDSL SIPROC4
-EXIT
```

If XL.PUB._youracct_ already exists, add the SUPERDEX modules into it from the file XL.PUB.SUPERDEX:

```
:LINKEDIT
-XL  yourxl
-COPYXL; XL.PUB.SUPERDEX
-EXIT
```

The SUPERDEX SL or XL may alternately reside in a group SL or XL (in a group other than PUB). Copy the SL or XL into the group in which the programs reside, or add the segments or modules into an existing SL or XL as shown above.

☎ If using BASIC/3000, please call Bradmark Technologies for instructions. If using Business Basic, refer to the supplementary documentation for the *Business Basic Interface*.

If you don't want to run your programs via an SL or XL, the SUPERDEX procedures may alternately be included in program files during the PREP or the LINK.

Creating a group for utility programs

Copy the utilities (QUERY, DICTDBL, etc.) and other programs (TRANSACT, etc.) you use for database transactions into a group in an account containing the SUPERDEX SL or XL (not PUB.SYS). The recommended method is to create a new **SUPERDEX** group in the **SYS** account, copy the utilities into it, and run them with **;LIB=G**. You may want to rename or lockword the copies of the utilities that remain in PUB.SYS to prevent accidental use which could result in a loss of synchronization between the SI-indices and entries.

Adding capabilities to accounts and groups

SUPERDEX unconditionally requires **DS** capability, and optionally **MR** capability; DS capability is not required under MPE/XL for groups/accounts that contain only Native Mode programs. MR capability is required for programs that have SUPERDEX automatically lock its internal dataset(s)--refer to the discussion in the *Locking* chapter of the *Programming* section to determine if MR capability is necessary for your installation.

If required, add DS and optionally MR capabilities to groups and accounts in which programs that access SUPERDEX'ed databases reside:

```
:ALTACCT account; CAP=current capabilities,DS,MR
:ALTGROUP group; CAP=current capabilities,DS,MR
```

☞ All programs, except those in Native Mode, that access the SUPERDEX SL--even those which do not utilize SUPERDEX capabilities--require DS capability.

Adding capabilities to users

Programmers who will :PREP source programs which use SUPERDEX will need DS and optionally MR capabilities added to their users, unless they are :LINKing only Native Mode programs under MPE/XL, which do not require DS capability:

```
:ALTUSER userid; CAP=current capabilities,DS,MR
```

Adding capabilities to programs

Programs that use SUPERDEX, including utility programs like QUERY, must also have DS and MR capabilities. Several methods are available:

- use ALTPROG.PUB.SUPERDEX
- if you have VESOFT's MPEX/3000 utility, use the **ALTFILE** command (e.g. **ALTFILE @.@.account (CODE="PROG"); CAP=+DS,+MR)**
- re-:PREP with **CAP=**current capabilities**,DS,MR**

Adding stack space to programs

All SUPERDEX procedures automatically extend a program's stack by up to two Kwords for indexed access and one or more additional Kwords for relational access (if performed), so programs which run short of stack may abort with a STACK OVERFLOW. For programs in which this occurs, increase the program's MAXDATA using one of the methods below. If the MAXDATA is already at its upper limit, run the program ;NOCB:

- use ALTPROG.PUB.SUPERDEX
- use MPEX/3000 **ALTFILE** command (e.g. **ALTFILE program; MAXDATA=**(current maxdata + 2048))
- re-:PREP with **MAXDATA=**(current maxdata + 2048)

Changing menus, UDCs, and job streams

All application and utility programs that use SUPERDEX capabilities must be run with ;LIB=P or ;LIB=G, depending on whether the SUPERDEX SL or XL are in an account or group SL or XL, respectively:

```
:RUN program; LIB=P
```

Native Mode programs on MPE/XL may alternately be run by:

```
:RUN program; XL="XL.PUB.SUPERDEX"
```

All menus, UDCs, job streams, and other facilities that run these programs must be changed, and users who run programs explicitly from MPE must be instructed to include ;LIB=P or ;LIB=G on the :RUN command.

Expanding system tables

On MPE/V systems, the LOADPROC intrinsic called by SUPERDEX utilizes entries in two system tables, which may require expansion. The tables are:

- LOADER SEGMENT TABLE
- SWAP TABLE

Additionally on MPE/V systems, SUPERDEX creates one or more Extra Data Segments (XDS), which may require that the system's virtual memory allocation be increased.

☞ Refer to HP's **System Operation and Resource Management Reference Manual** for information about these tables and how to change them.

Configuration overview

Defining SI-keys and SI-paths

SI-paths are defined in the **SIMAINT** program, which initially creates the B-tree structures and optionally the SI-item and SI-dataset(s).

The SIMAINT program may need to know any of the following, which are discussed in detail on the following pages:

- whether a separate DBOPEN should be done for each database through which to perform all locking and unlocking of indices
- whether to maintain the SI-indices in the base with the data entries or in a separate base
- the dataset to which each SI-path is related
- the SI-dataset that contains the SI-indices for a dataset's SI-paths
- the name of each SI-path
- the field or fields that form each SI-key and their lengths
- whether each SI-path is keyworded, grouped, both, or neither
- whether to index entries that contain blank values in the first SI-subkey, for each SI-path
- the keyword length, for each keyworded SI-path
- the minimum number of characters per keyword, for each keyworded SI-path
- the average number of significant words that will be contained in the SI-key, for each keyworded SI-path
- the keyword exclusion length, for all keyworded SI-paths
- the SI-key length, for each custom SI-path
- the SI-index length, for each independent SI-path
- up to four special characters to optionally exclude as keyword delimiters
- an optional replacement character for the @ wildcard/terminator
- an optional replacement character for the ? matchcode operator
- an optional replacement character for the # matchcode operator
- whether to disable multiple indexing of hyphenated values for keyworded SI-paths

Separate DBOPEN for locking

SUPERDEX is capable of implicitly locking the appropriate SI-dataset whenever a DBPUT, DBUPDATE, or DBDELETE is performed. Because IMAGE has no selective unlock facility, the lock on the SI-dataset is held until the program calls DBUNLOCK. This could impede throughput with multiple users.

For this reason, SUPERDEX may be configured to DBOPEN a database twice for each process and perform its SI-dataset locks via the second DBOPEN. It is then able to unlock the SI-dataset via that open rather than waiting for the program to call DBUNLOCK.

To configure SUPERDEX to perform a separate DBOPEN for locking, append **/2** to the database name.

Separate database for SI-indices

By default, SUPERDEX's B-tree structures reside in the SI-dataset(s) in the same database as the data entries that they index. However, it is possible to alternately locate the SI-indices in a separate database, referred to as an *SI-index base*. In either case, the SI-definitions must reside in the root SI-dataset in the same database as the data entries (the *primary* base).

The main advantage to keeping the SI-indices in a separate database is that fewer log records are written for bases enabled for logging. Because SUPERDEX uses IMAGE intrinsics for maintaining its SI-indices, additional log records are generated, causing log files to fill more quickly. This situation becomes even more apparent with multiple indexing (multiple SI-indices for a single data entry), as required for keywording. By locating the SI-indices in a separate base, no additional logging activity is generated for the primary base.

Logging may be selectively enabled or disabled for the SI-index base. In either case, it is always possible to recover the SI-indices. If logging is enabled, logging recovery may be used in the event of a failure. If disabled, the inconsistent SI-paths may be re-indexed as necessary by reorganization using the SIMAINT utility program.

With this option, the root SI-dataset and SI-item reside in the primary database and contain only the SI-definitions. All SI-indices reside in one or more SI-datasets in the separate SI-index base.

This feature may be used selectively for each database. First, create the SI-index base in the same group and account as the primary base, and assign it the same name as the primary base but with the last two characters **SI**. If the primary base name is 4 characters or less, append **SI** to the base name; if 5 or 6 characters, replace the last one or two characters with **SI**. For example, the SI-index base for the OEDB base would be **OEDBSI**, and for CUSTDB would be **CUSTSI**.

The SI-index base may contain additional items and datasets in addition to the SI-dataset(s). The first SI-dataset must be named **SI1**, with additional SI-datasets named **SI2 - SI7**. Each SI-dataset must have the same block size and blocking factor as the root SI-dataset in the primary base.

☞ Refer to the *Internal structures* appendix for more information about the required characteristics of the SI-index base.

Once the SI-item and root SI-dataset have been added to the primary base and the SI-index base has been created, proceed to configure the SI-paths using SIMAINT, and append **/3** to the database name.

☞ This feature and the separate DBOPEN for locking feature (configured by **/2**) are mutually exclusive, since all locking against the SI-index base is always done independently—via a separate DBOPEN against each base.

Naming SI-paths

The name of the SI-path is important because it is later used in the *item* parameter of SUPERDEX's DBFIND intrinsic. Also, it may not contain a forward slash (/) unless the SI-path name is identical to an item name that contains a forward slash.

The SI-path name must be unique within a dataset but multiple datasets may contain SI-paths of the same name, unless the SI-path is for a super-group, in which case the SI-path name must be unique within the database. It is recommended that the SI-path name not be the same as the item name that forms the SI-key or any other item, since this can cause programs that are attempting IMAGE access via the IMAGE path to instead perform SUPERDEX access via the SI-path. This may, however, be desirable, as explained below.

If an SI-path related to a dataset is given the same name as an existing item used as an IMAGE search field in the same dataset, the SI-path is used instead of the IMAGE path. If the entry cannot be found using the SI-path, the IMAGE path is used instead. If both fail, condition word 17 ("**NO ENTRY**") is returned. This is useful for replacing automatic master sets with SI-paths: just name the SI-path the same name as the search field in the detail dataset and programs do not require modification.

☞ **Replacing an automatic master dataset with one or more SI-paths causes additional log records to be written for databases enabled for logging.**

If, however, you would like to access entries alternately by an SI-path and by an IMAGE path, assign a name other than the item name to the SI-path.

Simple vs. concatenated SI-keys and their lengths

An SI-key can be *simple* (a single field) or *concatenated* (a combination of fields). The latter is useful for both searching and sorting by extended criteria.

A concatenated SI-key for a detail set may consist of up to four fields. For a master set, the search field and up to two additional fields may be defined. If more than four fields are required in a concatenated SI-key and the desired fields are contiguous, they may be included and thereby exceed the four field limit. If the search field is included as an SI-subkey in a concatenated SI-key for a master set, it must be defined last.

For a simple SI-key or for each SI-subkey in a concatenated SI-key, each alphanumeric item that exceeds one word may be included in full, or its length may be shortened (and can start in any position). This is referred to as a substring SI-key or substring SI-subkey. The exception is master dataset search fields used in SI-keys, which may not be substring.

☞ When performing relational access against multiple datasets, it is required that a common item (called an *SI-link*) from each dataset be included in an SI-key for each dataset, except for master datasets in which the common item is the search field. The recommended method for implementing this is to include the common item as an SI-subkey in a concatenated SI-key.

Compound items

Compound IMAGE items may be used in simple SI-keys or as the first SI-subkey in concatenated SI-keys, including compound items with subitems of odd-byte lengths. SUPERDEX automatically treats compound items as grouped SI-keys and generates a separate SI-index for each subitem value, the result being that every subitem is always searched automatically.

For concatenated SI-keys that include compound IMAGE items, an SI-index is automatically generated for each subitem value, with the values of the other SI-subkeys in the SI-key repeated in each SI-index. For example, for an SI-key that consists of the compound item ORDER-COMMENTS (a 5X72) and the regular item ORDER-NUMBER (an X12) five SI-indices would be created with each one containing a different ORDER-COMMENT value but the same ORDER-NUMBER.

Keyworded SI-paths

Any *alphanumeric* (data type U or X) SI-key, either simple or concatenated, may be defined as keyworded. For concatenated SI-keys, only the first SI-subkey is keyworded.

Keyworded SI-paths are configured in SIMAINT by appending **/K** to the SI-path name. Three attributes must be specified for each keyworded SI-path:

- maximum length of each keyword
- minimum number of characters per keyword
- average number of keywords in the field for each entry

The *keyword length* determines the maximum number of characters to include in the SI-key, which is independent of the length of the field. For indexing purposes, words that are longer than the keyword length are truncated; those that are shorter are padded with spaces. For example, with a keyword length of 5 words, the first 10 characters of each word would be included, so for the word "MANUFACTURING," only "MANUFACTUR" would be included in the SI-key.

☞ **It is desirable for efficiency to keep the keyword length as short as possible and not to exceed a keyword length of 6 words (12 characters).** Substring keywords can still be retrieved using the full keyword as the search *argument*, but other entries may also be returned; for example, **MANUFACTURING** would qualify entries with the value "MANUFACTURER" because only 10 characters are stored in the SI-index and the argument length is substringed to 10 characters.

The *minimum number of characters per keyword* determines the minimum number of characters that a word must contain in order to qualify for keywording. For example, the word "ASK" would be included in keywording with a minimum keyword length of 1, 2, or 3 but excluded with a length of 4. The minimum keyword length may be between 1 and 4, with 1 effectively meaning that all words are included in keywording. This value should be set to 4 wherever possible for efficiency.

The *average number of keywords* refers to the average number of significant keywords that would be contained in this field, between 1 and 16. For example, "ACME MANUFACTURING PARTS" contains three keywords. The value of this parameter is used to reserve sufficient internal space, so fractional averages must be rounded up, and it is better to estimate high instead of low if in doubt; for example, if an average of 2.5 words are contained in the field, specify a value of 3. If requirements change at a later time, the average number of indices may be changed by reorganizing the SI-path--a new value may be specified at that time.

To eliminate common words from keywording, an exclusion list may be defined, which restricts the keyword entries to only relevant words. Exclusion words are specified in a disk file using any editor and uploaded into a special standalone SI-path called **KWEXCLUDE**. You must define a keyword length for this SI-path, which is applied against all the keyworded SI-paths in the database. The keyword length may be redefined at a later time if required.

Grouped SI-paths

Multiple SI-keys related to a given dataset may be grouped together as a single SI-path, with the following restrictions:

- each SI-key in the group must be of the same data type
- each SI-key in the group must internally have the same length, which is assigned for the first configured SI-key in the group and inherited by subsequently-configured SI-keys

For concatenated SI-keys in a group:

- the second - *n*th SI-subkeys are automatically and unconditionally imposed on all SI-paths in the group
- the first SI-subkey of every SI-key in the group must be of the same data type
- the first SI-subkey of each SI-key must be assigned the same length

Because each SI-key in a group must be of the same length, it is possible that the lengths of some SI-keys must be truncated and others padded with spaces. For example, if COMPANY is X30, CONTACT is X20, and PHONE is X14, these three fields may be grouped together with any length between 7 and 15 words--with 7 words, the SI-key for both COMPANY and CONTACT will be truncated; with 15 words, both CONTACT and PHONE would be padded with spaces; or any length in between could be chosen.

SI-paths are grouped together as a configuration option in SIMAINT by appending /G to each SI-path name except the first. The length and second - *n*th SI-subkeys of the first SI-path defined are unconditionally applied to subsequent SI-paths belonging to the same group, so define the SI-path that contains the desired SI-subkeys and longest SI-key first. If one of the SI-keys in a group for a master dataset is the IMAGE search field, define it last.

Super-grouped SI-paths

A master set and one or more related detail sets--related by IMAGE paths--may be super-grouped together as a single SI-path, with the following restriction:

- the item name of the search field used to form the IMAGE path must be the same

For concatenated SI-keys in a super-group:

- the second - *n*th SI-subkeys are automatically and unconditionally imposed on all SI-paths in the super-group
- the first SI-subkey of every SI-key in the super-group must be of the same data type
- the first SI-subkey of each SI-key must be assigned the same length

It is not required that a field in the master set be configured as an SI-key: it is possible to configure only SI-keys in the detail sets to be used to qualify entries in the related master set. In this case, use the field in the detail set when configuring the master path. This will be used to define the length and type of path.

SI-paths are super-grouped together as a configuration option in SIMAINT by appending /G to each SI-path name except the SI-path related to the master set. The length and second - *n*th SI-subkeys of the first SI-path defined are unconditionally applied to subsequent SI-paths belonging to the same super-group.

Custom SI-paths

Custom indexing (maintained by the SIUSER procedure) is implemented by defining an SI-path of an arbitrary name for each custom index. Although the dataset is defined, along with the SI-path, no items are specified, since the SI-indices do not directly reference any items. The SI-key length, excluding the SI-extension, is defined, as well as the average number of SI-indices per entry.

Independent SI-paths

Independent indexing is implemented by configuring a standalone B-tree for each independent SI-path. The dataset is left blank, and the SI-path name and SI-index length, including the extension, are defined.

Blank SI-keys

By default, SUPERDEX will not generate any SI-indices for any entry that contains a blank SI-key value. For a concatenated SI-key, it will not generate any SI-indices for an entry whose first SI-subkey is blank.

This is done for efficiency and disk savings, but differs from IMAGE's method of creating a "null" chain of all blank keys. To override SUPERDEX's default and cause SI-indices to be generated for entries with blank SI-keys, append /B to the SI-path name.

Summary of restrictions

Most configuration options may be used in combination against all data items, although some restrictions exist. This summarizes the aforementioned restrictions:

- only alphanumeric items (data types U and X) may be keyworded
- keywording functions only for the first SI-subkey in a concatenated SI-key
- substring fields may only be defined for alphanumeric items (data types U and X) whose lengths exceed one word. Master set search fields and numeric items are always represented in full and may not be truncated
- all SI-keys in a group must be of the same data type
- all SI-keys in a group must be assigned the same length (alphanumeric fields may be truncated or padded with spaces)
- if a grouped SI-path for a master dataset contains the IMAGE master search field, it must be configured last
- concatenated SI-keys may contain a compound item as the first SI-subkey only

Excluding words from keywording

Definition and Purpose

To minimize disk utilization, SUPERDEX permits common words to be excluded from keywording. These exclusion words are user-specified and apply to all keyworded SI-paths in a database.

All the words that SUPERDEX should exclude from keywording for a database are entered into a file named **KWEXCLUD** and then transferred into the special Independent SI-path **KWEXCLUDE** by the SIMAINT program.

Just as each keyworded SI-path has an associated keyword length that determines how many characters of each keyword are recognized for indexing, the KWEXCLUDE SI-path also has an associated *keyword length* which is applied against each word in the exclusion file. Because the exclusion words are compared against all keyworded SI-paths, and because each keyworded SI-path may have a different keyword length, it is important to assure that the excluded words will work effectively for each SI-path. This means that only as many characters as are defined by the keyword length are matched on; therefore, it is recommended that the shortest keyword length configured for any SI-path in the database be used as the keyword length for the KWEXCLUDE path.

Exclusion words are entered into a file named KWEXCLUD which must be located in the group/account in which the corresponding database resides. Multiple KWEXCLUD files may be used, one per database. If multiple databases reside in the same group/account and require different KWEXCLUD files, create additional files under different names and reference them with :FILE equations, for example:

```
:FILE KWEXCLUD=OEDBKW.DEMO.SUPERDEX
```

Example

Exclusion words may be entered in any editor that creates a plain ASCII file, one per line, and must be in upper case, as shown in the following example. (In this example, EDIT/3000 is used, but any editor that produces a standard ASCII file may be used.)

```
:EDITOR
HP32201A.07.17 EDIT/3000
(C) Hewlett-Packard CO. 1985
/A
    1       INC
    2       INCORPORATED
    3       CORP
    4       CORPORATION
    5       DIVISION
    6       ASSOC
    7       ASSOCIATES
    8       //
/K KWEXCLUD
/EXIT

END OF SUBSYSTEM
```

For this example, the keyword length that will be imposed on the KWEXCLUDE SI-path (when it is defined in SIMAINT) is 4 words (8 characters). The words on line 2 and 4 exceed this length, so they will be truncated. The shorter words on lines 1, 3, and 6 will be padded with spaces, which are ignored for comparison.

Once the KWEXCLUD file has been created, it is necessary to define the KWEXCLUDE SI-path using the SIMAINT program, as described in the following chapter. Although this file may be created and modified at any time, it is recommended that it be created in its entirety before configuring any keyworded SI-paths because any changes to the file require that the KWEXCLUDE path and all keyworded SI-paths be reorganized.

Default File

The KWEXCLUD.DEMO.SUPERDEX file is a default file containing common words to be excluded. Except in very special cases, it is recommended that this file be used in databases with keyworded SI-paths to reduce the amount of D. For example:

```
:FILE KWEXCLUD=KWEXCLUD.DEMO.SUPERDEX
:RUN SIMAINT.PUB.SUPERDEX,LIST

SIMAINT VERSION 3.1(06DEC91) COPYRIGHT DR. MATT / IABG (1988,1991)

DATABASE >OEDB
DATASET > space + return
SI-PATH > KWEXCLUDE
```

Customizing default characters

SUPERDEX recognizes a few special characters that influence the operation of various intrinsics. Because of unique characteristics that your data may have, SUPERDEX allows you to redefine these characters to suit your individual requirements.

For DBFIND, SUPERDEX reserves the following special characters as conditional operators:

- @ as wildcard and terminator
- ? as alphanumeric matchcode
- # as numeric matchcode

For DBPUT, DBUPDATE, DBDELETE, and SIMAINT indexing on keyworded SI-paths, the following special characters are reserved:

- all special characters as keyword delimiters
- − as a keyword delimiter which results in multiple indexing of hyphenated values

So, if your system contains data values that include ?s or #s as literal characters, it would be restrictive to use the default ? or # character as a single-character matchcode; therefore, the matchcode operator could be redefined to some other character not commonly found in your data values (such as %).

☞ **When assigning replacement characters for @, #, and ?, it is important to choose characters other than those already treated specially by DBFIND as relational or Boolean operators (documented under DBFIND in the _Intrinsics_ section in this manual).**

Another circumstance in which default characters should be redefined is in recognizing keyword delimiters for keyworded SI-paths. By default, spaces and all special characters are treated as keyword delimiters, and it may be desirable to restrict which special characters are recognized. For example, values that include fractions ("1/2") or dates ("01/30/90") suggest that / should be excluded as a keyword delimiter; otherwise "1/2" would be indexed as "1" and "2."

Additionally, because hyphenated values are by default indexed multiple times ("01-30-90" is indexed as "01-30," "30-90," and "90") it may be desirable to disable this feature and instead treat hyphens (−) as regular keyword delimiters (resulting in "01," "30," and "90"). Or, it may be best to disable hyphens as keyword delimiters altogether.

To redefine default characters, include a customization string of up to eight characters which defines the desired defaults as an INFO parameter when running the SIMAINT program (described later). The wildcard/terminator, alphanumeric matchcode, numeric matchcode, and up to four excluded keyword delimiters can all be included in the customization string.

The customization string specified when running SIMAINT affects all SI-paths in the database.

☞ **Once specified, the customization string is written into the internal SI-definitions for the database and need not be specified again.** If the default characters need to be changed at a later time, run SIMAINT with a new customization string; if keyword delimiters are changed, also reorganize all keyworded SI-paths.

To preserve the default values for wildcard and matchcode operators (@, #, and ?) it is necessary to specify them explicitly in the customization string.

The eight characters (bytes) in the customization string are represented as follows:

byte	description
1	wildcard and terminator character
2	alphanumeric matchcode, blank disables this feature
3	numeric matchcode; blank disables this feature
4	− if multi-indexing on hyphen, blank to treat hyphen as regular delimiter
5	special character to be treated as literal rather than keyword delimiter
6	same
7	same
8	same

For example, running SIMAINT with the customization string shown:

```
:RUN SIMAINT.PUB.SUPERDEX;INFO="%^* /:-."
```

would result in the following: % is recognized (in place of @) as the wildcard and terminator character, ^ (instead of ?) is the single-character matchcode, * replaces # as the single-numeric matchcode, no multi-indexing is performed for hyphenated values, and /, :, −, and . are not treated as keyword delimiters but rather as regular literal characters.

To display the configured customization string, run SIMAINT with the LIST entry point as shown:

```
:RUN SIMAINT.PUB.SUPERDEX,LIST

SIMAINT VERSION 3.1(06DEC91>COPYRIGHT DR. MATT / IABG (1988,1991)

DATABASE >OEDB
CUSTOMIZATION STRING: %^* /:-.
THE FOLLOWING SI-PATHS AND ITEMS ARE DEFINED:
. . .
```

Configuring SUPERDEX using SIMAINT

The SIMAINT program automatically establishes the B-tree structures and SI-indices for all SI-paths when they are created. It adds the SI-item and SI-dataset(s) for TurboIMAGE and TurboIMAGE/XL databases, unless a separate SI-index base is used. For non-Turbo bases, use DBGENERAL, or DBUNLOAD/DBLOAD to establish the SI-dataset(s) and SI-item. For separate SI-index bases, use the DBSCHEMA and DBUTIL utilities to create the SI-index base, or use the SIBASE utility program *(see Section 6)*.

(SIMAINT is also used for deleting, reorganizing, and performing other maintenance functions on existing SI-paths. Options for existing SI-paths are not discussed in this section but in the *Maintenance and utilities* section, which also includes a table describing the various operations that may require SI-path maintenance.)

Creating SI-item and SI-dataset(s)

The SI-item and SI-dataset(s) that are used for storing SUPERDEX's B-trees may be created by one of two methods:

- by SIMAINT.PUB.SUPERDEX, which creates them automatically
- by conventional means, such as DBGENERAL, or DBUNLOAD/DBLOAD

The preferred method is to use SIMAINT, since it easier and automatic. After creating the SI-item and SI-dataset(s) based on the SI-path configuration, SIMAINT proceeds to create the required SI-definitions and SI-indices; however, the following requirements must be met in order to use SIMAINT.PUB to create the SI-item and SI-dataset(s):

- the system must be running TurboIMAGE or TurboIMAGE/XL (not IMAGE)
- all dataset block sizes must be even multiples of 128 words (e.g. 128, 256, 384, 512), which is normally the case
- the SI-indices must reside in the same database as the data entries (not in a separate SI-index base)

If the above conditions are not satisfied, instead run SIMAINT.NOPRIV which (based on the SI-path configuration) displays the characteristics of the SI-item and required capacities for the SI-dataset(s) but does not generate the SI-definitions or SI-indices. Use this information to manually create the SI-item and SI-dataset(s), referring to the *Internal structures* appendix for more information. Then, stream the job created by SIMAINT.PUB,SCHEMA to configure the SI-paths, and the SI-definitions and SI-indices will be generated.

Operation

The SIMAINT program is prompt-driven, and includes an on-line help facility that displays datasets and the items they contain. SIMAINT operates in three discrete phases, which are automatically invoked in succession:

- *dialog* phase: all configuration information is specified
- *extension* phase: the SI-item and SI-dataset(s) are added, as necessary
- *indexing* phase: all SI-indices are generated

This permits all configuration information to be specified up front in the dialog phase and for the program to be left unattended during extension and indexing.

Access requirements

Before running SIMAINT, make sure:

- you have exclusive access to the database
- you are logged on as the database creator
- you are logged into the group and account in which the database resides

It is also recommended for performance reasons that you:

- disable ILR
- disable logging
- do not run SIMAINT with ;LIB=G or ;LIB=P

Input rules

These rules govern SIMAINT input:

- all input may be in upper- or lower-case
- ? displays structural help (sets and items)
- \ flushes all activity for a given level and returns to the previous level
- space returns to the previous level in the hierarchy while retaining the activity in a level
- all lengths--with the exception of *minimum keyword length, average number of characters per keyword* and *offset in bytes,* which are always specified in bytes--are reported and specified in words if a positive value, and bytes if a negative value. It is necessary to convert for alphanumeric (data types U and X) items (e.g. X20 = 20 bytes or 10 words).

Dialog phase

SIMAINT dialog is structured in a hierarchical fashion whereby you are led down through various levels until all required information has been supplied, and then automatically returned to the previous level. The levels in the hierarchy are:

```
DATABASE >
   DATASET >
      SI-PATH >
         ITEM n >
            SI-subkey prompts
```

This organization encourages a logical ordering in configuring SUPERDEX for multiple datasets and items, assures that all required information is specified, and permits an easy and consistent method for canceling and reentering input for any level in the hierarchy.

SIEXTLEN JCW for special concatenated SI-keys

A concatenated SI-key can normally contain no more than four fields as SI-subkeys. It is possible, however, to create a concatenated SI-key with more than four SI-subkeys, provided that the additional fields are contiguous in the dataset. If they are, it is possible to define a length for any SI-subkey that exceeds the field length, and the additional fields or truncated fields that follow are included in the SI-key, based on the length specified.

If you need to utilize this feature, it is necessary to set a special JCW named **SIEXTLEN** to **1** before invoking SIMAINT.

```
:SETJCW SIEXTLEN=1
```

Invoking SIMAINT

To invoke SIMAINT for TurboIMAGE and TurboIMAGE/XL databases:

```
:RUN SIMAINT.PUB.SUPERDEX

SIMAINT VERSION 3.1(06DEC91) COPYRIGHT DR. MATT / IABG (1988,1991)
```

☞ SIMAINT is run with neither ;LIB=G nor ;LIB=P.

For IMAGE databases, as well as databases with block sizes that are not even multiples of 128 words (128, 256, 384, 512, etc.) instead **:RUN SIMAINT.NOPRIV.SUPERDEX** after creating the SI-dataset(s) and SI-item, refer to the *Internal structure* appendix for more information.

Defining database

SIMAINT can be run against databases that have never been configured for SUPERDEX as well as those that already contain SI-paths.

The OEDB database in these examples contains no SI-paths. Because the version of SIMAINT being run is SIMAINT.PUB, a message is displayed that the SI-detail dataset(s) will be added automatically:

```
DATABASE >OEDB
DATABASE HAS NOT BEEN INITIALIZED FOR SUPERDEX
AFTER THE DIALOG PHASE THE PROGRAM WILL GO INTO PRIVILEGED MODE TO ADD THE SI-
DATASET(S) TO THE DATABASE
```

Defining number of DBOPENs

In this example, the database is being configured for SUPERDEX to perform a second DBOPEN through which all locking of the SI-dataset(s) will be performed:

```
DATABASE >OEDB/2
```

☞ **/2** appended to the database name means that a second DBOPEN should be performed. If, instead, only a single DBOPEN should be done, with all locking done via that DBOPEN, either nothing or **/1** should be appended. The specified suffix is automatically retained for all subsequent configuration against the database. Either suffix may also be declared for an already-configured database, and SUPERDEX will override the old specification.

Defining that a separate SI-index base is used

In this example, the database being configured for SUPERDEX will contain the root SI-dataset but not the SI-dataset(s) that hold the SI-indices, rather, they will be maintained in a separate SI-index database:

```
DATABASE >OEDB/3
```

☞ **/3** appended to the database name means that SUPERDEX should maintain its SI-indices in a separate SI-index base, **which must already exist**. The suffix is automatically retained for all subsequent configuration and **may not be overridden.**

The **/3** option and the **/2** option may not be used together--they are mutually exclusive.

Defining datasets

SIMAINT can be run against datasets that already have related SI-paths as well as those that do not. If a dataset already contains SI-paths, they are displayed.

Enter the name of a manual master or detail dataset in the current database, optionally followed by one of the following suffixes:

/n SI-dataset to contain the SI-indices for all dataset's SI-paths, where n is the number of the SI-dataset between 1 and 7

/D Delete all dataset's related SI-paths (refer to *Maintenance and utilities* section)

/R Reorganize all dataset's related SI-paths (refer to *Maintenance and utilities* section)

In the following example, no suffix was specified, therefore all indices for SI-paths related to the PRFD dataset will reside in the root SI-dataset:

```
DATASET >PRFD
```

Once all datasets have been defined, hit RETURN at the **DATASET** prompt. This indicates to SIMAINT that you are done defining SI-paths, and causes it to proceed to the extension stage.

☞ **Once an SI-dataset has been selected for a dataset, all indices for that dataset must remain in the same SI-dataset. Different SI-paths for the same dataset cannot be placed in different SI-datasets.**

Defining associated SI-datasets

In this example, all the SI-indices for all SI-paths related to the CUST dataset will be built in the SI-dataset named **SI1**:

```
DATASET >CUST/1
```

Up to eight SI-datasets (including the root SI-dataset) may be configured for any database. Multiple SI-datasets are useful for optimizing concurrent access and necessary for large databases in which an SI-dataset may outgrow the maximum file size allowed by the MPE operating system (about 2 million sectors for MPE/V and 8 million sectors for MPE/XL). Valid SI-dataset suffixes are **/1** through **/7**. In assigning suffixes, do not skip any numbers.

The suffix is automatically retained for all subsequent configuration against the dataset, and may only be changed by deleting and re-adding all the SI-paths related to the dataset. Likewise, if the dataset was originally configured without a suffix, no suffix may be specified without redefining all the dataset's SI-paths.

☞ **With super-grouping the master and the related detail sets must be configured with the same SI-dataset.**

Bypassing datasets for independent SI-paths

When defining independent SI-paths which are not associated with any dataset, and when defining the keyword exclusion path **KWEXCLUDE**, a SPACE followed by a RETURN is specified at the dataset prompt:

```
DATASET >space + return
```

To specify which SI-dataset will contain all independent SI-paths for the database, specify the SI-dataset number in place of the SPACE, as shown:

```
DATASET >/2
```

Defining SI-paths and SI-keys

As described previously, SI-paths may be:

- simple or concatenated
- keyworded
- a grouping of multiple SI-keys
- customized, for SI-keys generated via the SIUSER procedure
- standalone, for independent indexing

As already discussed, each SI-path name is arbitrary, and may be the same as an existing item name. The SI-path name may optionally be appended by one of the following suffixes:

- **/K** SI-path is Keyworded
- **/G** SI-path consists of multiple SI-keys that form a Group
- **/B** Blank entries are indexed (by default, no SI-index is generated for an SI-key value that is blank or for a concatenated SI-key, whose first SI-subkey is blank)
- **/D** Delete existing SI-path (refer to the *Maintenance and utilities* section)
- **/R** Reorganize existing SI-path (refer to the *Maintenance and utilities* section)

After the SI-path name has been entered, SIMAINT prompts for the item(s) that comprise the corresponding SI-key. A simple SI-key consists of only one item; a concatenated SI-key may contain between two and four items.

For alphanumeric items (data types U or X), a truncated field rather than the full field may be included in both simple and concatenated SI-keys. For numeric items, no truncation is allowed, so the request to shorten the length is not asked.

Defining simple SI-keys

In this example, the arbitrary name *CITY* is assigned for the SI-path that consists of the full field CUCITY:

```
SI-PATH >CITY/B
ITEM 1 >CUCITY
ITEM LENGTH IS 8 WORDS.ENTER SHORTER LENGTH (- =BYTES) IF DESIRED>return
ITEM 2 >return
SI-PATH >
```

☞ RETURN was hit to accept the full field length for the SI-key and also hit when SIMAINT prompted for the second SI-subkey, thereby defining this SI-key as simple rather than concatenated. Also, /B was appended to the SI-PATH, which causes SUPERDEX to override its default and generate SI-indices for entries that contain blanks in the first SI-subkey.

If the name specified for an SI-path is the name of a field in the dataset--such as when replacing an automatic master dataset with an SI-path--SIMAINT assumes that the first SI-subkey is that item and therefore does not prompt for it:

```
SI-PATH >PRFIDP
ITEM LENGTH IS 2 WORDS.ENTER SHORTER LENGTH (- =BYTES) IF DESIRED>return
ITEM 2 >return
SI-PATH >
```

If the current dataset is a master and the SI-path name specified is the same as the dataset's search field, neither the item nor length is prompted for, since truncated SI-keys that are master set search fields are not allowed:

```
SI-PATH >CUCNUM
SI-PATH >
```

Defining concatenated SI-keys

To define a concatenated SI-key, specify additional fields to be included as SI-subkeys:

```
SI-PATH >ORDER-ID/B
ITEM 1 >OMCPON
ITEM LENGTH IS 7 WORDS. ENTER SHORTER LENGTH (- =BYTES) IF DESIRED >2
ENTER OFFSET IN BYTES (RETURN=1) > return
ITEM 2 >OMBLOC
ITEM 3 >OMCCOD
SI-PATH >
```

☞ There are several items to notice in this example. First, only the first 2 words (4 characters) of the OMCPON field are being included in the SI-key. Second, since the item has been shortened, the desired offset is requested. Next, the length was not prompted for in the second and third SI-subkeys because the items are not of an alphanumeric data type and numeric items must be represented in full. Fourth, only three SI-subkeys (rather than four) were prompted for because the SI-path is being configured for a master dataset and the IMAGE search field is not included in the SI-key. Lastly, /B was specified to cause SUPERDEX to generate SI-indices for entries that contain a blank value in the first SI-subkey.

The following example defines the same concatenated SI-path, but defines the length in bytes, instead of words. Additionally, an offset of 3 has been chosen.

```
SI-PATH >ORDER-ID/B
ITEM 1 >OMCPON
ITEM LENGTH IS 7 WORDS. ENTER SHORTER LENGTH (- =BYTES) IF DESIRED >-4
ENTER OFFSET IN BYTES (RETURN=1) > 3
ITEM 2 >OMBLOC
ITEM 3 >OMCCOD
SI-PATH >
```

In the following example, a concatenated SI-key containing three SI-subkeys is being configured for a detail dataset. **No SI-subkey lengths are prompted for because all items are numeric:**

```
SI-PATH >ITEM-ID
ITEM 1 >OMNUMB
ITEM 2 >ITMNUM
ITEM 3 >ITMCOD
ITEM 4 >return
SI-PATH >
```

☞ As explained earlier, a concatenated SI-key can contain more than four SI-subkeys only if the desired excess fields are contiguous in the dataset and if the special **SIEXTLEN JCW** is set to 1.

This example shows a concatenated SI-key containing eight SI-subkeys, although only four are referenced by name. The situation is that there are eight separate fields that hold status codes-- named *STATUS-1* through *STATUS-8*; and it is necessary to have all eight four-character codes included in the concatenated SI-key:

```
SI-PATH >STATUS
ITEM 1 >STATUS-1
ITEM LENGTH IS 2 WORDS. ENTER NEW LENGTH (- =BYTES) IF DESIRED >return
ITEM 2 >STATUS-2
ITEM LENGTH IS 2 WORDS. ENTER NEW LENGTH >return
ITEM 3 >STATUS-3
ITEM LENGTH IS 2 WORDS. ENTER NEW LENGTH >return
ITEM 3 >STATUS-4
ITEM LENGTH IS 2 WORDS. ENTER NEW LENGTH >10
ENTER OFFSET IN BYTES (RETURN=1) >return
SI-PATH >
```

☞ The default length of two words was chosen for the first three status codes (by hitting RETURN). A length of 10 was specified for *STATUS-4*, which will cause the remaining status codes (*STATUS-5* through *STATUS-8*) to be implicitly included in the SI-key. Also, the question displayed is not the same when the **SIEXTLEN JCW** is turned on.

Remember that in order to configure this special type of concatenated SI-key that it is necessary to have the **SIEXTLEN** JCW set to 1 before invoking SIMAINT.

Defining keyworded SI-paths

Appending **/K** to the SI-path name defines an SI-path that is keyworded:

```
SI-PATH >CUNAME/K
ITEM LENGTH IS 15 WORDS. ENTER KEYWORD LENGTH >6
ENTER MINIMUM NUMBER OF CHARACTERS PER KEYWORD (1-4) >4
ENTER AVERAGE NUMBER OF KEYWORDS PER ENTRY >3
ITEM 2 >return
SI-PATH >
```

☞ Instead of prompting for the SI-key length, SIMAINT instead prompts for keyword length, minimum number of characters per keyword, and average number of keywords. The keyword length can be entered in bytes as a negative value. Also, the offset cannot be specified. The entire field is included in the keyword.

In this example, each keyword is recognized only by the first 12 characters (6 words), a word must contain at least 4 characters to be keyworded, and an average of 3 keywords are contained in each SI-key.

Concatenated SI-keys can also be configured as keyworded, but only the first SI-subkey is keyworded:

```
SI-PATH >SHIP-ID/K
ITEM 1 >SHNAME
ITEM LENGTH IS 15 WORDS. ENTER KEYWORD LENGTH >-12
ENTER MINIMUM NUMBER OF CHARACTERS PER KEYWORD (1-4) >3
ENTER AVERAGE NUMBER OF KEYWORDS PER ENTRY >4
ITEM 2 >SHADD1
ITEM 3 >SHADD2
ITEM 4 >return
SI-PATH >
```

The keyword length of -12 (12 characters or 6 words), minimum number of characters per keyword of 3, and average number of keywords of 4 are automatically applied to the first SI-subkey (the other SI-subkeys are not keyworded).

Defining grouped SI-paths

A grouped SI-path is defined by specifying multiple SI-keys under a common SI-path name. It is not necessary to configure all members of the group at the same time: additional SI-keys may be added into the group at a later time (when deleting a grouped SI-path, however, all members are deleted).

These are the steps to follow in defining a grouped SI-path:

1. Specify a new SI-path name, optionally suffixed with /B (to index all blank SI-keys in the group) or /K (to define all SI-keys in the group as keyworded). The suffix attribute is inherited by all SI-keys in the group.
2. Specify the name of the first item to be contained as an SI-key in the group. Define the longest item first, since the SI-key length for the group may not exceed the length of the first item specified. If the group is related to a master dataset and includes the IMAGE search field, define it last.
3. Specify the SI-key length, which will be applied to all SI-keys in the group and which therefore must be no greater than the longest item in the group. SI-keys for items that are shorter than this length are padded with spaces.
4. Define additional SI-subkeys, if desired, when prompted. These SI-subkeys are unconditionally applied to all SI-keys in the group.
5. When prompted for the next SI-path name, specify the same SI-path name as before but append the suffix /G.
6. Enter the name of the next SI-key to be included in the group.
7. Repeat steps 5 and 6 for each additional SI-key in the group.

This example shows a grouping of two SI-key fields under the SI-path name *SHADD*:

```
SI-PATH >SHADD
ITEM 1 >SHADD1
ITEM LENGTH IS 13 WORDS. ENTER SHORTER LENGTH (- =BYTES) IF DESIRED >-10
ITEM 2 >return
SI-PATH >SHADD/G
ITEM 1 >SHADD2
SI-PATH >
```

☞ The first field defined is automatically included in the group, even though /G was not appended to the corresponding SI-path name.

This example shows a grouping of three fields under the SI-path *CUPHN*:

```
SI-PATH >CUPHN
ITEM 1 >CUPHN1
ITEM 2 >return
SI-PATH >CUPHN/G
ITEM 1 >CUPHN2
SI-PATH >CUPHN/G
ITEM 1 >CUPHN3
SI-PATH >
```

The SI-key length was not prompted for here because all items are integers (data type I).

Defining super-grouped SI-paths

A super-grouped SI-path is defined by configuring an SI-path for a master dataset and then referencing that SI-path for one or more related detail datasets. It is not necessary to include all related detail datasets in the super-group.

The SI-set number option on the datasets, both master and details, must be the same if a super-group is wanted.

SI-path names between masters and their related details should be uniquely named to avoid any confusion between super-grouped paths, and those that are not super-grouped.

☞ The items selected must exist in all datasets involved except for the first item on a master dataset.

These are the steps to follow in defining a super-grouped SI-path:

1. First configure a new SI-path for the master dataset, optionally suffixed with **/B** (to index all blank SI-keys in the group) or **/K** (to define all SI-keys in the group as keyworded). The suffix attribute is inherited by all SI-keys in the super-group.
2. Specify the name of the first item in the master set to be contained as an SI-key in the super-group. If no item in the master set is to be included in the super-group, specify the item from one of the related detail sets that will be part of the super-group.
3. Define additional SI-subkeys in the master set, if desired, when prompted. These SI-subkeys are unconditionally applied to all SI-keys in the super-group. If defining additional SI-subkeys for the master path, each subkey must also exist in the detail dataset(s), except for SI-subkey 1.
4. When prompted for the next SI-path name, hit **RETURN** to be prompted for the next dataset (or continue to define additional, unrelated SI-paths for the master set).
5. When prompted for the next dataset, specify the name of a detail dataset related by an IMAGE path to the master set previously specified.
6. When prompted for SI-path name, enter the same SI-path name as for the master set but append the suffix **/G**.
7. Enter the name of the item in the detail dataset that is to be included as an SI-key in the super-group.
8. Enter **RETURN** for SI-path name (or continue to define additional, unrelated SI-paths for that detail set).
9. Repeat steps 5 through 9 for each additional related detail datasets in the super-group.

This example shows a super-grouping of a master dataset and a single associated detail set (related by the IMAGE path along CUSTOMER-NUMBER) under the SI-path name *CUSTOMER-BRANCH*:

```
DATASET >CUSTOMERS
SI-PATH >CUSTOMER-BRANCH
ITEM 1 >CUSTOMER-NAME
ITEM LENGTH IS 15 WORDS.  ENTER NEW LENGTH (- =BYTES) IF DESIRED >return
ITEM 2 >return
SI-PATH >return
DATASET >BRANCHES
SI-PATH >CUSTOMER-BRANCH/G
ITEM 1 >BRANCH-NAME
SI-PATH >return
DATASET >
```

This example shows a super-grouping of the ORDER-LINES detail dataset that is related to the ORDER-HEADERS master dataset in which the master dataset is not included in the super-group, under the SI-path name *PART-DESCRIP*:

```
DATASET >ORDER-HEADERS
SI-PATH >PART-DESCRIP
ITEM 1 >PART-DESCRIPTION     << does not exist in ORDER-HEADERS but in ORDER-LINES >>
ITEM LENGTH IS 13 WORDS.  ENTER NEW LENGTH (- =BYTES) IF DESIRED >return
ITEM 2 >return
SI-PATH >return
DATASET >ORDER-LINES
SI-PATH >PART-DESCRIP/G
ITEM 1 >PART-DESCRIPTION
SI-PATH >return
DATASET >
```

Defining SI-paths that are both keyworded and grouped

As shown above, a grouped SI-path may be defined as keyworded, in which case every SI-key in the group is treated as keyworded.

The SI-path in this example is both keyworded and grouped, as defined simply by appending /K to the SI-path name when first declared:

```
SI-PATH >CUADD/K
ITEM 1 >CUADD1
ITEM LENGTH IS 13 WORDS. ENTER KEYWORD LENGTH >5
ENTER MINIMUM NUMBER OF CHARACTERS PER KEYWORD (1-4) >1
ENTER AVERAGE NUMBER OF KEYWORDS PER ENTRY >4
ITEM 2 >return
SI-PATH >CUADD/G
ITEM 1 >CUADD2
SI-PATH >
```

Defining custom SI-paths

For custom SI-paths, for which the SI-key is generated by the SIUSER procedure, specify an arbitrary SI-path name and RETURN for the first SI-subkey:

```
DATASET >SHIP
SI-PATH >CUSTOM-ID
ITEM 1 >return
ENTER SI-KEY LENGTH >4
ENTER AVERAGE NUMBER OF INDICES PER ENTRY >4
SI-PATH >
```

The *SI-key length* represents the length, **excluding** the SI-extension, that is returned in the *indices* parameter of the SIUSER procedure. The *average number of indices* represents the average value of the index count returned in the first word of the *index* parameter of SIUSER.

If the database is enabled for NLS (Native Language Support), the following question is also asked:

```
USE NLS-SORTING (N/Y) >
```

This determines whether NLS collating sequences are used for ordering entries for this SI-path. Because numeric data is sorted the same, respond **N** if the SI-path contains only numeric data fields; otherwise respond **Y**.

Defining independent SI-paths

For independent SI-paths that are not associated with any dataset, enter a **SPACE** and **RETURN** for the dataset and an arbitrary SI-path name:

```
DATASET >space + return
SI-PATH >DOCUMENT-NAME
ENTER SI-INDEX LENGTH >10
SI-PATH >
```

The *SI-index length* represents the length of the entire SI-index: both the SI-key and SI-extension. **This must be at least 2 words long.**

If the database is enabled for NLS (Native Language Support), the following question is also asked:

```
USE NLS-SORTING (N/Y) >
```

This determines whether NLS collating sequences are used for ordering entries for this SI-path. Because numeric data is sorted the same, respond **N** if the SI-path contains only numeric data fields; otherwise respond **Y**.

Defining keyword exclusion SI-path

The special independent SI-path named **KWEXCLUDE** is reserved for words excluded from keywording (as defined in the file KWEXCLUD, which should already exist in the same group/account as the database). Enter **SPACE** for the dataset and **KWEXCLUDE** for the SI-path:

```
DATASET > space + return
SI-PATH >KWEXCLUDE
ENTER KEYWORD LENGTH >5
SI-PATH >
```

The *keyword length* represents the significant length of each exclusion word. Refer to the *Excluding words from keywording* chapter in this section for a discussion.

Deferring indexing

By default, SIMAINT will proceed to the extension phase (if required) and indexing phase once all configuration information has been specified, as indicated by a response of **RETURN** at the DATASET> prompt. You may instead want to defer indexing until a more convenient time, since it does require exclusive database access.

To facilitate this, SIMAINT is capable of completing the extension phase (if required) and saving the newly-configured SI-definitions but deferring the indexing phase until explicitly specified. This is accomplished by responding **/N** to the DATASET> prompt, as shown:

```
DATASET>/N
```

To proceed with indexing when convenient, rerun SIMAINT against the same database and hit RETURN at the **DATASET>** prompt.

Extension phase -- specifying SI-dataset(s) capacity

Once all definitions have been entered, SIMAINT proceeds to calculate the capacities for every SI-dataset. The calculated SI-dataset capacities are worst-case calculations which are sufficient for storing the SI-indices that are required based on the current capacities of the datasets they represent. (Refer to the _Internal Structures_ appendix for more information about how the SI-dataset capacity is calculated.)

If SIMAINT.PUB is being run, the capacities are displayed and the opportunity is given to override them. The program then proceeds to add the SI-item and SI-dataset(s) into the database.

```
CALCULATED CAPACITY OF SI-DATASET 241
DESIRED CAPACITY (RETURN = CALCULATED CAPACITY) >400

CALCULATED CAPACITY OF SI1-DATASET 160
DESIRED CAPACITY (RETURN = CALCULATED CAPACITY) >250
EXTENSION STARTED - DO NOT INTERRUPT
EXTENSION SUCCESSFUL
```

With SIMAINT.PUB, the configuration process is now complete and the program proceeds to create the SI-indices.

If SIMAINT.NOPRIV is being run, the SI-item characteristics and calculated SI-dataset capacities are displayed and a job stream is created to facilitate configuration once the SI-item and SI-dataset(s) have been added:

```
CAPACITY OF SI-DATASET
========================
THE CALCULATION IS BASED ON THE CURRENT BLOCKMAX OF 512 WORDS
CREATE SI-ITEM AS FOLLOWS 4X254  (508 WORDS)
CREATE SI-DATASET WITH CAPACITY OF 241
CREATE SI1-DATASET WITH CAPACITY OF 160
```

With SIMAINT.NOPRIV, it is now necessary for you to manually create the SI-item and SI-dataset(s), as described in the *Internal structures* appendix. The SI-item should be created with the characteristics shown, and the SI-dataset capacity(ies) should be set at least as high as the recommended capacity(ies). Once this is done, stream the job created by SIMAINT to configure and create the SI-paths.

If you have configured SUPERDEX to utilize a separate SI-index base, use SIMAINT.NOPRIV to calculate the capacities for the SI-dataset(s) but do not append **/3** to the database name. Once the capacities have been determined, create the SI-index base as described in the *Internal structures* appendix.

Now, having created the SI-item and SI-dataset(s) and perhaps the SI-index base, stream the job created by SIMAINT.NOPRIV to actually configure the SI-paths. If using a separate SI-index base, append **/3** to the primary database name when re-running SIMAINT.

Indexing phase -- progress display

Once the extension is completed, SIMAINT proceeds to generate the SI-path definitions and SI-indices. Extensive progress reporting is displayed during the indexing phase.

Each non-independent SI-path must go through three processes. First, the records in the dataset must be read, called the INPUT phase. Next the created indices for the SI-path must be sorted, called the SORT phase. Finally the indices must be written out to the SI dataset, called the OUTPUT phase.

During the INPUT and OUTPUT phases a display of the number of entries read, or indices added, the percentage complete, the elapsed time, and the elapsed CPU seconds are displayed at a regular predefined interval. During the SORT phase, no progress reporting can be displayed until after the SORT is completed.

```
PROCESSING SI-PATH KWEXCLUDE       OF
PROCESSING SI-PATH CUSTOMER-NAME   OF CUSTOMERS  # OF ENT: 1003
     INPUT:    1003 RECORDS   100 %  CPU 0:00:01.2 Elapsed 0:00:05
     SORT:     1003 INDICES          CPU 0:00:00.0 Elapsed 0:00:00
     OUTPUT:   1003 INDICES   100 %  CPU 0:00:00.9 Elapsed 0:00:02
PROCESSING SI-PATH CUSTOMER-NAME-KW OF CUSTOMERS  # OF ENT: 1003
     INPUT:    1003 RECORDS   100 %  CPU 0:00:02.7 Elapsed 0:00:06
     SORT:     2803 INDICES          CPU 0:00:00.0 Elapsed 0:00:00
     OUTPUT:   2788 INDICES   100 %  CPU 0:00:01.6 Elapsed 0:00:03
PROCESSING SI-PATH ADDRESS1-CITY-KW OF CUSTOMERS  # OF ENT: 1003
     INPUT:    1003 RECORDS   100 %  CPU 0:00:04.3 Elapsed 0:00:07
     SORT:     4448 INDICES          CPU 0:00:00.0 Elapsed 0:00:00
     OUTPUT:   4417 INDICES   100 %  CPU 0:00:02.6 Elapsed 0:00:03
TOTAL TIME:                         CPU 0:00:17.7 Elapsed 0:00:40

END OF PROGRAM
```

☞ Notice that the number of records read during the INPUT phase was always the same as the # OF **ENT:**, but the number of indices sorted and written out either match the number of entries or is larger. This is because in an SI-path that is not a one-to-one path (only simple paths or simple-concatenated paths are one-to-one paths) will create more indices than number of records.

The number of indices written out may be less than then number sorted because all duplicate SI-indices are not added. In other words, if a record has the same word in a keyworded SI-path only one SI-index will be added.

The default interval for updating the progress report is 1000 records or indices. This can be overridden by setting the JCW **SICOUNT**. The valid values for this JCW are 100 to 32767, inclusive. If the value is not valid, then 1000 will be used.

Example of configuring and establishing SI-indices for a database

A complete example of an entire SIMAINT configuration session for a database follows:

```
:RUN SIMAINT.PUB.SUPERDEX
SIMAINT VERSION 3.1 (06DEC91) COPYRIGHT DR. MATT / IABG (1988,1991)

DATABASE > OEDB
DATABASE HAS NOT BEEN INITIALIZED FOR SUPERDEX
AFTER THE DIALOG PHASE THE PROGRAM WILL GO INTO PRIVILEGED MODE TO ADD
THE SI-DATASET(S) TO THE DATABASE
DATASET > space + return
SI-PATH > KWEXCLUDE
ENTER KEYWORD LENGTH > 4
SI-PATH > return
DATASET > CUSTOMERS
SI-PATH > CUSTOMER-NAME
ITEM LENGTH IS 15 WORDS.ENTER SHORTER LENGTH (- =BYTES)IF DESIRED> return
ITEM 2 > return
SI-PATH > CUSTOMER-NAME-KW/K
ITEM 1 > CUSTOMER-NAME
ITEM LENGTH IS 15 WORDS. ENTER KEYWORD LENGTH > 4
ENTER MINIMUM NUMBER OF CHARACTERS PER KEYWORD > 1
ENTER AVERAGE NUMBER OF KEYWORDS PER ENTRY > 6
ITEM 2 > return
SI-PATH > ADDRESS1-CITY-KW/K
ITEM 1 > ADDRESS-1
ITEM LENGTH IS 13 WORDS. ENTER KEYWORD LENGTH > 4
ENTER MINIMUM NUMBER OF CHARACTERS PER KEYWORD > 1
ENTER AVERAGE NUMBER OF KEYWORDS PER ENTRY > 4
ITEM 2 > return
SI-PATH > ADDRESS1-CITY-KW/G
ITEM 1 > CITY
SI-PATH > return
DATASET > ORDER-HEADERS
SI-PATH > CUSTOMER-NUMBER
ITEM 2 > return
SI-PATH > return
DATASET > ORDER-LINES
SI-PATH > ORDER-PART
ITEM 1 > ORDER-NUMBER
ITEM 2 > PART-NUMBER
ITEM LENGTH IS 7 WORDS.ENTER SHORTER LENGTH (- =BYTES)IF DESIRED > return
ITEM 3 > return
SI-PATH > PART-ORDER
ITEM 1 > PART-NUMBER
ITEM LENGTH IS 7 WORDS.ENTER SHORTER LENGTH (- =BYTES)IF DESIRED > return
ITEM 2 > ORDER-NUMBER
ITEM 3 > return
SI-PATH > return
DATASET > return
```

```
CALCULATED CAPACITY OF SI-DATASET 752
DESIRED CAPACITY (RETURN = CALCULATED CAPACITY) >1000
EXTENSION STARTED - DO NOT INTERRUPT
EXTENSION SUCCESSFUL

PROCESSING SI-PATH KWEXCLUDE        OF

PROCESSING SI-PATH CUSTOMER-NAME     OF CUSTOMERS         # OF ENT: 1003
          INPUT:   1003 RECORDS    100 %   CPU 0:00:01.3 Elapsed 0:00:02
          SORT:    1003 INDICES            CPU 0:00:00.0 Elapsed 0:00:00
          OUTPUT:  1003 INDICES    100 %   CPU 0:00:00.9 Elapsed 0:00:01
PROCESSING SI-PATH CUSTOMER-NAME-KW OF CUSTOMERS         # OF ENT: 1003
          INPUT:   1003 RECORDS    100 %   CPU 0:00:03.0 Elapsed 0:00:03
          SORT:    3046 INDICES            CPU 0:00:00.0 Elapsed 0:00:00
          OUTPUT:  3032 INDICES    100 %   CPU 0:00:01.8 Elapsed 0:00:02
PROCESSING SI-PATH ADDRESS1-CITY-KW OF CUSTOMERS         # OF ENT: 1003
          INPUT:   1003 RECORDS    100 %   CPU 0:00:04.6 Elapsed 0:00:05
          SORT:    4375 INDICES            CPU 0:00:00.0 Elapsed 0:00:00
          OUTPUT:  4344 INDICES    100 %   CPU 0:00:02.6 Elapsed 0:00:03
PROCESSING SI-PATH CUSTOMER-NUMBER  OF ORDER-HEADERS     # OF ENT: 2620
          INPUT:   2620 RECORDS    100 %   CPU 0:00:02.4 Elapsed 0:00:03
          SORT:    2620 INDICES            CPU 0:00:00.0 Elapsed 0:00:00
          OUTPUT:  2620 INDICES    100 %   CPU 0:00:01.4 Elapsed 0:00:02
PROCESSING SI-PATH ORDER-PART       OF ORDER-LINES       # OF ENT: 9272
          INPUT:   9272 RECORDS    100 %   CPU 0:00:09.1 Elapsed 0:00:10
          SORT:    9272 INDICES            CPU 0:00:00.1 Elapsed 0:00:00
          OUTPUT:  9272 INDICES    100 %   CPU 0:00:06.9 Elapsed 0:00:09
PROCESSING SI-PATH PART-ORDER       OF ORDER-LINES       # OF ENT: 9272
          INPUT:   9272 RECORDS    100 %   CPU 0:00:09.1 Elapsed 0:00:10
          SORT:    9272 INDICES            CPU 0:00:00.1 Elapsed 0:00:00
          OUTPUT:  9272 INDICES    100 %   CPU 0:00:07.3 Elapsed 0:00:09
TOTAL TIME :                                CPU 0:00:57.8 Elapsed 0:06:27

END OF PROGRAM
```

Running SIMAINT in batch

SIMAINT can be run in batch, and uses dialog similar to on-line. The method for creating a job stream by which to run SIMAINT in batch is to build a job file using the SCHEMA entry-point after all paths have been defined.

The discrepancies between on-line and batch use are:

- all prompts are displayed during batch, while during on-line the prompts are variable depending on answers
- a line containing only a SPACE is represented in batch by a blank line
- a line containing only a RETURN (which is normally specified in a batch job as a blank line) is represented by a line containing a double slash (//) in the first two character positions

SIMAINT will QUIT (not TERMINATE) normally upon encountering any error in batch, permitting testing of the system JCW.

Running SIMAINT with a ;STDIN option will cause SIMAINT to execute as if in batch.

Overview

This section discusses the various methods utilized in programming with SUPERDEX and gives examples of various types of SUPERDEX access.

Although the SUPERDEX intrinsics are discussed throughout this section, they are documented fully for reference in the *Intrinsics* section.

Chapter 1   **Locking**
Function   in this section examines various *Locking* strategies available in SUPERDEX, both explicit and implicit, as well as multiple DBOPENs and other locking considerations.

Chapter 2   **Adding, updating, and deleting entries**
Function   discusses *Adding, updating, and deleting entries* using DBPUT, DBUPDATE, and DBDELETE. Also covered are custom indices, the DBPUTIX and DBDELIX intrinsics, and independent SI-paths.

Chapter 3   **Qualifying entries with DBFIND**
Function   describes various methods of *Qualifying entries with DBFIND*. All the various access methods are shown, including keyword and grouped retrieval, as well as lookups that involve a single set, multiple SI-paths in a single set, multiple sets, and multiple bases.

Chapter 4   **Retrieving entries with DBGET**
Function   looks at *Retrieving entries with DBGET* in sorted sequential order using *modes* 5, 6, 15, and 16, as well as methods for reading masters and their related details and SI-indices only.

Chapter 5   **Additional programming considerations**
Function   examines *Additional programming considerations*, such as :PREParing programs and programming language variations.

Chapter 6   **Native Language Support**
Function   describes special considerations that must be made when using HP's *Native Language Support* facility, for adding, updating, and indexing entries and qualifying entries with DBFIND.

Locking

Because the SI-indices may require changes with every DBPUT, DBUPDATE, DBDELETE, DBPUTIX, and DBDELIX, it is necessary to lock the SI-dataset(s) whenever these intrinsics are called against SUPERDEX'ed datasets and the database is DBOPENed in *mode* 1.

SUPERDEX has several strategies for locking the SI-dataset(s) when necessary:

- explicit locking of <u>all</u> datasets, including the SI-dataset(s)
- implicit locking of all the datasets that are not explicitly locked (entry datasets as well as SI-dataset(s))
- implicit locking of the SI-dataset(s) only
- implicit locking and unlocking of the SI-dataset(s) via a separate DBOPEN of the base
- implicit locking and unlocking of the SI-dataset(s) contained in a separate SI-index database via a separate DBOPEN of the SI-index base

Explicit locking of all datasets

If you do not want SUPERDEX to perform any implicit locking, add the SI-dataset(s) to the program's lock descriptor, using an *item* of @.

Implicit locking of all the datasets

SUPERDEX is capable of automatically performing <u>all</u> required locking (both DBLOCKs and DBUNLOCKs) in any program against all the datasets in a database, including those that do not have associated SI-paths. In order to utilize this method, the program must not lock the datasets that have related SI-paths.

When an application calls DBPUT, DBUPDATE, or DBDELETE with no covering lock, SUPERDEX automatically locks the dataset that contains the data entries, modifies it, and then unlocks it. It then locks the appropriate SI-dataset, modifies it, and then unlocks it. Both locks are applied at the dataset level.

This method is very efficient but because the lock against the SI-dataset is applied at the dataset level, it is recommended that any locking performed by programs be done at the dataset level to avoid waiting for item-level locks.

Implicit locking of the SI-dataset(s)

Existing programs that already lock do not need to be modified for SUPERDEX. SUPERDEX is capable of automatically performing a set-level lock against the appropriate SI-dataset when required.

This method requires that SUPERDEX lock an SI-dataset when another lock against the dataset containing the entries is being held. To accomplish this, MPE requires that the program and the group and account in which it resides have MR (multi-rin) capability. (By granting MR capability, it is possible that an undetected conflicting lock inherent in a program will result in a deadlock, whereas the result without MR capability would be an IMAGE error.) A deadlock could occur, however, if a separate program performs a base-level lock during the window between the set-level lock against an entry dataset and the SI-dataset. If such a condition exists--with a separate program performing base-level locks concurrently with another program performing set-level locks against a dataset that contains SI-paths, either locking should be done explicitly or SUPERDEX should be configured to maintain its SI-indices in a separate SI-index base, as explained below.

SUPERDEX determines whether to implicitly lock the SI-dataset(s) by checking the lock descriptor when the program calls DBLOCK. If an SI-dataset (SI - SI7) is not included, SUPERDEX automatically performs a set-level lock against the appropriate SI-dataset. The lock is released when the program calls DBUNLOCK, unless SUPERDEX was configured to lock and unlock via a separate DBOPEN (as discussed below).

In this method, SUPERDEX locks only one SI-dataset, so if a logical transaction consists of multiple DBPUTs, DBUPDATEs, or DBDELETEs involving multiple datasets with SI-paths, all these datasets' SI-indices must be contained in the same SI-dataset in order to be covered by the implicit lock. If this is not the case, the program must explicitly lock all related SI-datasets; alternately, SUPERDEX may be configured to perform a separate DBOPEN or maintain its SI-indices in a separate SI-index base, as discussed below, to eliminate this requirement.

Implicit locking and unlocking of the SI-dataset(s) via separate DBOPEN

Because IMAGE does not have a selective unlock facility, the SI-dataset lock placed by SUPERDEX in the previous method is released when the program calls DBUNLOCK. This strategy could impede throughput of multiple database users if the delay between the DBPUT, DBUPDATE, or DBDELETE call and the DBUNLOCK call is significant.

☞ **It is recommended that SUPERDEX be configured to perform a separate DBOPEN for each process through which all SI-dataset locking will be done. With this method, all entry-dataset locks are done via the first DBOPEN, and all SI-dataset locks via the second DBOPEN. This permits SUPERDEX to call DBUNLOCK immediately after modifying its SI-indices, although the dataset lock placed via the first DBOPEN may still be held. An additional benefit is that logical transactions against multiple datasets whose SI-indices are in separate SI-datasets are locked for accordingly. This method requires MR capability.**

Implicit locking and unlocking of the SI-dataset(s) in SI-index base

If SUPERDEX has been configured to maintain all the SI-indices for a database in a separate SI-index base, a separate DBOPEN is automatically performed against the SI-index base and a set-level lock automatically applied to the appropriate SI-dataset in the SI-index base.

In this case, no SUPERDEX locking is required against any datasets in the primary base containing the data entries, since only the root SI-dataset exists and it is only read and not written.

This strategy is recommended for users who do not want the overhead of transaction logging of intrinsics used by SUPERDEX to maintain the SI-indices in addition to the regular logging of transactions against data entries. Using this method, logging may be either enabled or disabled on the SI-index base while enabled on the primary base.

Because programs must hold two locks simultaneously (against both the entry dataset in the primary base and an SI-index dataset in the SI-index base) this method requires MR capability.

Adding, updating, and deleting entries

Adding and deleting entries with DBPUT and DBDELETE

Entries are added and deleted using SUPERDEX's DBPUT and DBDELETE intrinsics, which are identical to their IMAGE counterparts except they additionally maintain the associated SI-indices.

For bases that are enabled for logging, SUPERDEX automatically imposes DBBEGINs and DBENDs around each DBPUT and DBDELETE that does not already specify them. The DBEND is imposed regardless of whether or not the DBPUT or DBDELETE is successful.

Updating entries with DBUPDATE

Entries are updated using SUPERDEX's DBUPDATE intrinsic, which is identical to its IMAGE counterpart except that a delete and add of any related SI-indices is internally performed.

If DBUPDATE specifies neither the @ *list* nor a *list* that includes all the SI-subkeys whose values are being changed, SUPERDEX will automatically perform one or more rereads (DBGET *mode* 1) with various *lists* when performing the DBUPDATE. This is both inefficient and causes the current *list* to change without the knowledge of the program, and therefore any subsequent calls done without re-initializing the *list* may be faulty. It is therefore recommended that the @ *list* or a *list* that contains all SI-subkeys be used when calling DBUPDATE.

In the event that DBUPDATE is performed against a database that was DBOPENed in *mode* 2, no SI-keys may be updated since this would require that a DBPUT and DBDELETE be performed against the corresponding SI-dataset, which may not be done in DBOPEN *mode* 2. If this is attempted, condition word -41 ("**DBUPDATE WILL NOT ALTER A SEARCH OR SORT ITEM**") is returned.

For bases that are enabled for logging, SUPERDEX automatically imposes DBBEGINs and DBENDs around each DBUPDATE that does not already specify them. The DBEND is imposed regardless of the success of the DBUPDATE.

DBGET used to locate entry for DBUPDATE or DBDELETE

If the DBGET used to locate an entry for updating or deletion specifies a *list* that does not include all SI-subkeys, SUPERDEX may internally change the *list*. The *list* will be changed for DBUPDATE if it does not include all SI-subkeys whose values are being changed; for DBDELETE, all SI-subkeys in the dataset must be included. Refer to the *Retrieving entries with DBGET* chapter later in this section for a discussion.

Determining SI-key value

Normally, SUPERDEX automatically determines, based on its configuration, the SI-key and SI-subkey values to include in the SI-indices.

There are instances, however, in which the SI-key value is not represented in the data entry in any straightforward manner and SUPERDEX is therefore unable to determine it. Some examples of this are as follows:

■ **Date reformatting**
   Date must be converted from *yy/mm/dd* format into Julian or other format

■ **Upshifting**
   SUPERDEX only upshifts SI-keys for SI-paths that are configured as keyworded. You may want to upshift SI-keys for non-keyworded SI-paths

■ **Compound indexing**
   SUPERDEX allows a maximum of four SI-subkeys in an SI-key, which may be insufficient

■ **SOUNDEX**
   A SOUNDEX system whereby sound-alike searching is possible may be desired

SUPERDEX provides two different facilities for addressing these needs: the **SIUSER** procedure, and the **DBPUTIX** and **DBDELIX** intrinsics.

Custom SI-indices with SIUSER

SIUSER is a user-written procedure that is used by SUPERDEX to compute one or more custom SI-indices for entries whenever DBPUT, DBUPDATE, or DBDELETE is called and whenever SIMAINT is run.

SIUSER requires that the database, dataset, and SI-path be defined in its *base, dset,* and *item* parameters and that the full data entry be supplied in its *buffer* parameter. SIUSER returns the number of SI-indices created and their values in the *index* parameter.

Explicit SI-index management with DBPUTIX and DBDELIX

The DBPUTIX and DBDELIX intrinsics explicitly add and delete SI-indices, and are useful for generating custom SI-indices in addition to those automatically generated by DBPUT and removed by DBDELETE.

The common method for establishing multiple SI-indices for an entry, over and above those generated by DBPUT, is to first call DBPUT to add the entry to the dataset and create the configured SI-indices, and then call DBPUTIX as many times as necessary to establish the additional SI-indices. The same technique is used with DBDELETE and DBDELIX to remove SI-indices.

DBPUTIX and DBDELIX require that the database, dataset, and SI-path be defined in their *base, dset,* and *item* parameters and that the full SI-index, including the SI-extension, be supplied in the *buffer* parameter.

For a master set, the SI-extension is the entry's IMAGE search field value, which for DBPUT may be retrieved from the *buffer* parameter and for DBDELETE may be gotten from the *list* used by the DBGET that located the entry for deletion. For a detail set, the SI-extension is the entry's relative record number, which is returned in words 3-4 of the *status* array from the DBPUT or DBDELETE.

Independent SI-paths

The DBPUTIX and DBDELIX intrinsics are also used for independent indexing, in which the entities being indexed do not reside in IMAGE datasets.

DBPUTIX and DBDELIX against independent SI-paths require that the *base* parameter identify the database in which the SI-indices are contained and the *dset* parameter be left blank or set to 200. The SI-path is defined in the *item* parameter, and the full SI-index, including an appropriate SI-extension, is provided in the *buffer* parameter.

Qualifying entries with DBFIND

In IMAGE, entries in a detail dataset are normally accessed by using DBFIND to locate the head of a chain (which is contained in a related master dataset) for a specified search field value (commonly called a "key value"), and then the associated entries are subsequently retrieved via DBGET *mode* 5 or 6. An entry in a master dataset is normally accessed via DBGET *mode* 7, based on the search field value specified.

SUPERDEX also uses DBFIND and DBGET to access dataset entries. The same techniques are used for both master and detail datasets in the same way.

In SUPERDEX, DBFIND does much more than locate the chain head for a specified key value: it qualifies multiple entries based on various criteria, and may be called multiple times in succession to refine the selection using various fields, datasets, and databases.

Once entries have been qualified with DBFIND, they may be retrieved in ascending or descending sorted sequential order with DBGET *mode* 5 and 6 or new DBGET *modes* 15 and 16.

Effectively, qualifying and retrieving entries in a master or detail dataset using an SI-path is just like qualifying and retrieving entries in a detail set using an IMAGE path.

Summary of DBFIND options

Whereas IMAGE's DBFIND works only on detail datasets, SUPERDEX's DBFIND works on:

- detail datasets
- master datasets
- SI-indices only, which may reference external files

Based on how SI-paths are configured, DBFIND can qualify entries by any of the following:

- any word in a keyworded SI-key
- a combination of SI-subkey values in a concatenated SI-key
- a value that occurs in multiple SI-keys that are grouped
- any subitem in a compound IMAGE item that is configured as an SI-key

DBFIND can qualify entries generically by any of the following, as specified in the *argument* by conditional and relational operators:

- start with a specified value (partial key)
- contain a specified value (embedded key)
- greater-than or equal-to a specified value
- less-than or equal-to a specified value
- not equal to a specified value
- in a range of two values

DBFIND also permits multiple generic or exactly-matching values to be specified in an *argument*, which permits the use of range and Boolean operations to find all qualifying entries:

- by an **AND** combination of multiple values
- by an **OR** combination of multiple values
- by an **AND NOT** combination of multiple values

In performing *relational access*, multiple successive DBFINDs may be called and their results ANDed, ORed, and AND NOTed by Boolean operators. This dynamically achieves the following operations:

- refine a selection with additional criteria
- undo a selection (revert to results of prior DBFIND)
- qualify by multiple SI-keys in a single dataset
- qualify by multiple datasets in a database
- qualify by multiple databases

New Relational Operator

A new option has been added for those situations where several relational DBFINDs are being executed and the relation tables overflow.

If a DBFIND contains an argument with two or more values to be OR'ed together, and then AND'ed to the pervious DBFIND, an overflow can occur even when it is possible that the net result of qualified records would fit in the table. Two new DBFIND operators have been added to make this process possible; **[ ]** and **$**. The **[ ]** will automatically copy the active SI-subset to the backup SI-subset and clear the active SI-subset. The **$** will then AND the search value to the backup SI-subset, placing the qualified entries in the active SI-subset.

The net result of this process is that the record qualified by the additional DBFIND will not automatically be written to the active SI-subset, but will first be qualified against the backup SI-subset. This means that the active SI-subset will never contain more records than the backup SI-subset, eliminating most table overflows.

To illustrate the process:

1. DBFIND on path NAME with argument **[A@]** places the records that qualified into the active SI-subset.

2. DBFIND with argument **[ ]** will replace the backup SI-subset with the active and create an empty active SI-subset.

3. DBFIND on path DATE with argument **[9111@]$[91212@]$** will process each selection separately. First all the records in the backup SI-subset with a DATE of **9111@** will be copied to the active SI-subset, then the records in the backup SI-subset with a DATE of **9112@** will be copied to the active SI-subset.

Steps 2 and 3 could be combined into a single DBFIND on path DATE with an argument of **[ ][91111@]$[9112@]$**.

Before these new operators, steps 2 and 3 would have been in a single DBFIND on path DATE with argument **[9111@][9112@] | &**.

Indexed access vs. relational access

Internally, one of two access methods is used in qualifying entries with DBFIND: *indexed access* or *relational access*.

With indexed accessed, DBFIND locates the first qualifying SI-index that matches the specified *argument* in the SI-path defined by the *item* parameter. With indexed access, the B-tree in the corresponding SI-dataset is accessed, and all qualifying entries form a logical SI-chain. Entries on the SI-chain are returned using DBGET by reading up or down the B-tree and retrieving the corresponding entries one-by-one.

With relational access, SI-indices are qualified using the same method but are then copied to form a virtual SI-chain in the *active SI-subset*. Entries are retrieved using DBGET by reading up or down this virtual SI-chain. The advantage is that unlike indexed access, the SI-chains contained in the active SI-subset are available for combination with entries located by subsequent DBFIND calls.

To recap, *indexed access* retrieves entries by directly reading a logical SI-chain in the SI-dataset; *relational access* first forms the logical SI-chain and then copies it to form a virtual SI-chain in the active SI-subset.

Indexed access is used for retrievals that can be accomplished by accessing a single SI-chain. This accounts for most retrievals, and is used by default.

Relational access is used for Boolean retrievals that require the use of multiple SI-chains, such as in performing retrievals against multiple SI-paths, sets, and bases by using multiple DBFIND calls. It is only invoked when explicitly specified by enclosing the *argument* value(s) in square brackets ([ ]), or enclosing the complete *argument* with tilde and semi-colon (˜ . . . ;).

DBFIND modes

Typically, most DBFINDs are performed using either *mode* 1 or *mode* 10. Either *mode* may be used for both indexed and relational access methods, although they are treated identically when performing relational access. In indexed access, both *modes* perform the same function, except *mode* 10 does not return the qualifying number of entries in the *status* array (always returns a count of 1) and may therefore be considerably more efficient than *mode* 1. *Mode* 10 should be used instead of *mode* 1 in indexed access whenever possible for efficiency.

Modes 1*nn* and 2*nn* in indexed access read *nn* words of the *argument* and set the SI-pointer before or after the first qualifying entry, respectively. If *nn* is prefixed by a minus sign (−), they read *nn* bytes instead of words. These *modes* are refinements and therefore do not handle as many *argument* constructs as *modes* 1 and 10, but are useful for certain circumstances, such as approximate match retrieval in which no qualifying value exists which sets the SI-pointer to the nearest qualifying entry. These *modes* are also useful in the unlikely event that a combination of symbols used to represent a SUPERDEX operator (e.g. >=) conflicts with a value in an SI-key in a data entry.

Modes 1*nn* and 2*nn* may also be used in relational access against a virtual SI-chain in the active SI-subset if a null *item* is specified. Refer to *Positioning on virtual SI-chain* later in this section for a discussion.

Modes **100** and **200** position the SI-pointer at the logical beginning and end of the dataset (lowest and highest entry in ascending sorted sequential order) or a virtual SI-chain in the active SI-subset, respectively.

DBFIND arguments used for indexed access

For indexed access, if DBFIND is called in *mode* 1 or 10 and the specified *argument* value is not the full SI-key value, either the *buffer* must be padded with spaces or the *argument* value must be terminated by:

- for alphanumeric fields, a single blank followed by a single @
- for alphanumeric fields, a single @ (used for partial-key retrieval)
- for numeric fields, a single blank

For DBFIND *modes* 1 and 10 in indexed access, ASCII numbers may be specified for most numeric items (data types I, J, P, R, and Z) if prefixed with ==, >=, <=, or <> or, if appropriate, a − (negative sign) -- the == operator simply converts an ASCII value specified to binary format for comparison.

☞ **The exception is data type K, for which values must be specified in binary.**

For concatenated SI-keys that contain SI-subkeys of mixed data types (alphanumeric vs. numeric) and for which the first SI-subkey is numeric, an ASCII value may be specified in *modes* 1 and 10 for the first SI-subkey only.

When performing relational access against numeric items, the *argument* must be specified in ASCII. |

For data types P and Z, SUPERDEX's DBFIND treats unsigned and positive values equivalently.

Real numbers (items of data type R) may include embedded decimal points (.), exponential signs (E), and positive (+) and negative (−) signs.

DBFIND mode/argument examples

Simple SI-key: alphanumeric X12

mode	argument	description
1	GOLDENBERG	exact match on "GOLDENBERG"
10	GOLDENBERG	same, but does not return the number of qualifying entries in the *status* array
1	GOLD@	all that start with "GOLD"
102	GOLD	same
1	G?LD	the values "GELD" and "GOLD"
1	G?LD@	the values "GELD," "GOLD," "GOLDEN," and "GOLDBERG"
1	?OLD	the values "BOLD," "COLD," "FOLD," and "GOLD"
1	PX??4400@	all that start with "PX," followed by any two characters, followed by 4400, followed by anything
1	PX??44@	all with "PX" followed by any two characters, followed by "44," followed by anything
1	>=A@<=B@	all in the range between "A" and "B," inclusive
1	<>87@	all except those that begin with "87"
100	(ignored)	the first alphabetical value in the dataset on the specified SI-path
200	(ignored)	the last alphabetical value in the dataset on the specified SI-path

Simple SI-key: numeric R2

mode	argument	description
1	>=1000	greater than or equal to 1000
1	>=123.4E5	greater than or equal to the specified number
1	<=100	less than or equal to 100
1	>=100<=1000	all in the range between 100 and 1000, inclusive

Simple SI-key: numeric Z4

mode	argument	description
1	1234	exact match on the value 1234
102	1234	same
-103	123	exact match on 123
any	0034	the value 34 (leading zeroes must be specified)
any	000J	the signed value -1 (leading zeroes must be specified)
1	>=1234	all greater than or equal to 1234
1	<=000J	all less than or equal to the signed value -1
1	>=23<=24	all in the range between 23 and 24 inclusive (leading zeroes need not be specified if prefixed with ==, >=, <=, or <>)

Concatenated SI-key: X4 + X4		
mode	*argument*	*description*
1	ABCD@	"ABCD" in the first SI-subkey, any value in the second SI-subkey
1	ABCD1234	exact match on both SI-subkeys: "ABCD" and "1234"
104	ABCD1234	same
-108	ABCD1234	same
1	????1234	"1234" in the second SI-subkey, any value in the first SI-subkey
10	????1234	same, but does not return the number of qualifying entries in the *status* array
-111	ABCD123456	condition word -31 ("BAD MODE"), because the SI-key length was exceeded

Concatenated SI-key: X4 + I1 (values underlined are specified in binary)		
mode	*argument*	*description*
-106	ABCD<u>1234</u>	exact match on "ABCD" in the first SI-subkey and 1234 in the second SI-subkey (where 1234 is specified in binary)
-105	ABCD1234	condition word -31 ("BAD MODE"), because I1 requires 2 bytes and therefore *mode* -106 should be used
1	ABCDE	all with "ABCD" in the first SI-subkey, and any value in the second SI-subkey (the extra character is ignored)
10	ABCDE	same, but does not return the qualifying number of entries in the *status* array

Concatenated SI-key: I1 + X4 (values underlined are specified in binary)		
mode	*argument*	*description*
103	<u>1234</u>ABCD	exact match on the value 1234 in the first SI-subkey and "ABCD" in the second SI-subkey (where 1234 is specified in binary)
-103	<u>1234</u>A	all with 1234 in the first SI-subkey and "A" leading the second SI-subkey (truncated value)
1	<u>1234</u>A	all with 1234 in the first SI-subkey (the extra character is ignored)
1	>=1234	all greater than or equal to 1234
1	<=2999	all less than or equal to 2999
1	>=-100<=100	all greater than or equal to -100 (negative 100) and less than or equal to 100 (positive 100)
-101	spaces	condition word -31 ("BAD MODE"), because the *mode* exceeds the argument length
1	1234	all equal to 12624 (first word, value "12", converted from ASCII value to binary value)

Concatenated SI-Key: Z4 + X4		
mode	argument	description
104	1234ABCD	exact match on the value 1234 in the first SI-subkey and "ABCD" in the second SI-subkey
-106	1234AB	exact match on 1234 in the first SI-subkey and "AB" in the second SI-subkey
1	1234	all with 1234 as the first SI-subkey
1	1234ABCD	same (the value for the second SI-subkey is ignored)
any	0034	the value 34 (leading zeroes must be specified)
1	0034ABCD	same (the value for the second SI-subkey is ignored)
1	>=1234ABCD	results unpredictable; any characters specified after the first SI-subkey renders the entire argument invalid
1	>=1234AB<=1234AB	same

Finding entries in a concatenated SI-key

In searching for entries using a concatenated SI-key, the entire concatenated value is considered. This permits an entry to be located by very specific criteria. For example, a concatenated SI-key comprised of an X2, X4, and X4 and containing the three SI-subkey values "PH," "1234," and "ABCD" would be located by:

   argument   = PH1234ABCD

If the concatenated SI-key is comprised of SI-subkeys that are all alphanumeric (data types X or U, as in the previous example), DBFIND modes 1 and 10 may be used to qualify entries using a full or partial SI-key value.

☞ If, however, the SI-subkeys are of both alphanumeric and numeric data types, modes 1nn and 2nn must be used. An exception is that if the leftmost SI-subkey is alphanumeric, a partial value appended with an @ may be specified to match on the leftmost alphanumeric SI-subkey(s).

Finding entries in a group

Searching for entries in multiple SI-keys in a dataset that are grouped together is completely transparent--DBFIND treats an SI-path containing a group of SI-keys as if it were a single SI-key and unconditionally searches all SI-keys in the group.

For example, if the items PHONE-1, PHONE-2, and PHONE-3 were grouped together in an SI-path, a phone number contained in any of the three fields would be searched for in a single DBFIND call.

For searches against an SI-key in a group which is shorter than the other SI-keys in the same group and is therefore padded with spaces, it is necessary to either pad the argument with spaces or perform a partial-key retrieval when calling DBFIND.

Finding entries in a super-group

Searching for entries in multiple SI-keys in multiple sets in a database that are super-grouped together qualifies master entries based on the contents of the SI-keys that form the super-group.

For example, if the book title is contained in the master dataset BOOK, the book author is contained in a related detail set AUTHOR, and the book summary is contained in the related detail set SUMMARY, and the three sets are super-grouped together based on book number, a specified value would be searched for in all three datasets. Master entries that contain the specified value in the title would qualify, as well as master entries that contain the specified word in either related detail dataset.

To qualify master entries, DBFIND is called against the master set specifying the SI-path name of the super-group in the item parameter, as shown:

dataset	= **BOOK**
SI-path	= **BOOK-KEY**
argument	= **PLAN@**

Finding entries in a compound IMAGE item

SI-keys that contain compound IMAGE items are automatically handled as if they were grouped; all subitems are always searched when the SI-path is referenced in the *item* parameter.

Finding entries by keyword

Searching for the occurrence of a keyword in an SI-path that has been configured as keyworded is completely transparent: the keyword is simply specified in the *argument* parameter.

If a given keyword occurs more than once in a single data entry, the entry is always returned only once. However, if multiple words in a single entry meet the search criteria, the entry will by default be returned multiple times. For example,

argument	= **PLAN@**

an entry containing the value "THE PLANNING COMMISSION'S NEW PLAN" would be returned twice. To prevent entries from being returned multiple times, specify relational access by placing a tilde (˜) in front of the "argument" and placing a semi-colon (;) at the end of the "argument".

˜PLAN;

☞ If the length of a keyword specified exceeds the keyword length configured for the referenced SI-path, the keyword value is truncated to the configured keyword length and matching is done based on the truncated value. Because of this, it is possible that entries with keywords that exceed the configured keyword length will be erroneously qualified. For example, a keyworded SI-key with a configured keyword length of 4 words (8 characters) containing the value "INDUSTRIOUS" would be qualified by an *argument* of "INDUSTRIAL," since only 8 characters are indexed and matched on.

Finding entries using a partial key

DBFIND can search for entries using a partial key value, with the @ character. The @ is treated as a wildcard (as in :LISTF) that stands for any number of any characters.

For example, to find all entries that begin with GENERAL":

    argument   = **GENERAL@**

The partial key value specified is compared with the entries.

The @ can be used up to two times within an *argument*. This is accomplished by surrounding the requested *argument* with **<< >>**. The format is **<<A@B@C>>**. Which means that this argument will qualify records that begin with an A, having a B anywhere in the middle, and ending with a C. If only one @ is provided, there is an implied @ at the end of the argument (e.g.**<<A@B>>** is identical to **<<A@B@>>**).

For example, to find all entries that begin with "GE" and contain "AL":

    argument   = **<<GE@AL>>**

☞ The @ can also be used as the first character in an argument. For example, **<<@OWN>>** will return BROWN and CROWN.

Finding entries using a generic key

The **?** and **#** characters facilitate generic searches. The **?** represents a single alphanumeric character and the **#** a single numeric character (as in :LISTF). They may occur multiple times anywhere in the value, for example:

    argument   = **STR?NG**

would locate "STRING," "STRONG," and "STRUNG."

    argument   = **AP#J3@**

would locate "AP1J379C," "AP8J3AQ4," and "AP4J3", but "APBJ3826" would not qualify.

Both matchcodes may be used in combination with each other, along with the @ wildcard, for example:

    argument   = **AP?J#@**

would additionally find "APBJ3826," "AP7J8AH," and "APZJ277." In this example, the @ acts as the terminator because the argument is not surrounded with **<< >>**.

The matchcodes may also be used to locate entries in which the desired value does not begin in the first position; for example:

    argument   = **??RT?N**

would locate "BARTON," "BURTON," "MARTIN," and "MORTON."

This technique is especially useful for generic searches on concatenated SI-subkeys, by specifying wildcard conditional operators for the unspecified SI-subkeys. These three examples perform generic searches on the first, second, and third SI-subkeys, respectively, of a concatenated SI-key consisting of an X2, X4, and X4 field with the value "PH1234ABCD":

argument   = **PH@**

argument   = **??1234@**

argument   = **??????ABCD**

☞ **Instead of specifying a ? at the beginning of an argument, it is more efficient to define and offset.**

Finding entries greater than or equal to a specified value

Greater-than-or-equal-to searches are accomplished using the **>=** relational operator to prefix the value. For example, to find all entries greater than or equal to 1000:

argument   = **>=1000**

SUPERDEX does not have a greater-than operator: **>** is not recognized and is therefore treated as a regular character. To accomplish a greater-than search, add one to the value being searched for:

argument   = **>=1001**

If the field being searched is of IMAGE data type R, specify the value in the following format:

argument   = **>=1000<>1000**

Finding entries less-than or equal-to a specified value

Less-than-or-equal-to searches are accomplished using the **<=** relational operator to prefix the value. For example, to find all entries less than or equal to 500:

argument   = **<=500**

SUPERDEX does not have a less-than operator: **<** is not recognized and is therefore treated as a regular character. To accomplish a less-than search, subtract one from the value being searched for:

argument   = **<=499**

If the field being searched is of IMAGE data type R, specify the value in the following format:

argument   = **<=1000<>1000**

Finding entries not equal to a specified value

To find entries not equal to a particular value, use the <> relational operator. For example, to find all unpaid orders:

argument   = <>PAID

The <> operator may also be embedded within an *argument* to perform a Boolean AND NOT retrieval. For example, to find all the entries with ZIP-CODES (an X6 item) beginning with "900" but not in "90039":

argument   = 900@<>90039

Finding entries in a range of values

The >= and <= relational operations may be used in combination to specify a range. For example, to find all the entries that start with letters between "A" and "D," inclusive:

argument   = >=A@<=D@

Pattern-matching and/or exclusion may optionally be performed within a range, allowing entries to be qualified that not only fall between two values but also conform to a specific pattern. For example:

argument   = ####AA>=8910@<=8912@

would find entries that fall between the values "8910" and "8912" and additionally contain "AA" as the fifth and sixth characters.

☞ The argument following >=, <= and <> may NOT contain embedded ? or # characters

DBFIND arguments used for relational access

For relational access, any of the previous *arguments* used for indexed access may be specified, and operate in the same way.

Unlike indexed access, however, search values specified in ASCII for retrieval against a binary field are automatically converted (and therefore do not need to be prefixed by the == conversion operator).

There are three ways to specify arguments for relational access: the SQL Notation, the Infix Notation, and the Reverse Polish Notation.

☞ The SQL Notation is the use of "**AND**", "**OR**", and "**NOT**" as Boolean operators to specify the relationship between the *argument* values. If the SQL Notation is to be used, the entire *argument* must begin with the tilde (˜) character and end with the semi-colon (;). The operators can be in either upper- or lower-case, including any combination of both. The *argument* must be left justified and contain no embedded spaces within each *argument* value unless the argument value is enclosed in quotes.

For example, an *argument* of `˜JONES @ OR SMITH@;` is not a valid argument.

SQL Notation Operators		
Oper	*Description*	*Format*
AND	AND'ed	~x **AND** y;
OR	OR'ed	~x **OR** y;
NOT	AND NOT'ed	~x **NOT** y;

☞ Infix Notation is the use of "**+**", "**,**", and "**-**" as Boolean operators. The use of Infix Notation also requires the entire *argument* to begin with the tilde (˜) and end with the semi-colon (;). There must not be any spaces within the entire *argument*. For example, the *argument* `˜JONES@, SMITH@;` is not valid. It must be `˜JONES@,SMITH@;`.

Infix Notation Operators		
Oper	*Description*	*Format*
+	AND'ed	~x+y;
,	OR'ed	~x,y;
-	AND NOT'ed	~x-y;

☞ When Reverse Polish Notation (RPN) is used, each *argument* value must be enclosed within square brackets ([ ]), and the entire *argument* must be terminated by a blank or @ (a value within square brackets needs no trailing character).

Reverse Polish Notation Operators		
Oper	*Description*	*Format*
&	AND'ed	[x] [y]&
|	OR'ed	[x] [y]	
!&	AND NOT'ed	[x] [y] !&

DBFIND mode/argument examples

Relational retrievals		
mode	argument	description
1	[GOLD@]	all that start with "GOLD"
1	~GOLD@;	same
1	[>=A@] [<=B@] &	all in the range between "A" and "B", inclusive
1	~>=A@ AND <=B@;	same
1	~>=A@+<=B@;	same
1	[A@] [B@]		all that begin with "A" or "B"
1	~A@ OR B@;	same
1	~A@,B@;	same
1	[>=A@] [AB@] !&	all greater than or equal to those that begin with "A", except those that begin with "AB"
1	~>=A@ NOT AB@;	same
1	~>=A@-AB@;	same

Finding entries by ANDing multiple values

A DBFIND *argument* may contain multiple values that are ANDed together with one of the AND Boolean operators (**AND**, **+**, **&**), and only those entries that qualify based on all specified values are selected.

For example, to find all the entries in a grouped SI-path that contain both the values "JOHN" and "CHICAGO":

argument	= ~CHICAGO AND JOHN;	SQL Notation
argument	= ~CHICAGO+JOHN;	Infix Notation
argument	= [CHICAGO] [JOHN] &	Reverse Polish Notation

☞ In the above example "CHICAGO" was specified first and "JOHN" second. This was done because "CHICAGO" appears on fewer entries than "JOHN". It is faster and more efficient to specify the less common value first when performing relational access retrievals.

Finding entries by ORing multiple values

Multiple values in a DBFIND *argument* may be ORed together using one of the OR Boolean operators (**OR**, **,** **|**), and entries that qualify based on any specified value are selected.

For example, to find all the entries in a keyworded SI-path that contain the word "FITTING," "NIBBLE," or "CONNECTOR":

argument	= ˜**FITTING OR NIBBLE OR CONNECTOR;**	SQL Notation
argument	= ˜**FITTING,NIBBLE,CONNECTOR;**	Infix Notation
argument	= **[FITTING][NIBBLE] | [CONNECTOR] |**	Reverse Polish Notation

Finding entries by AND NOTing multiple values

Multiple values in a DBFIND *argument* may be AND NOTed together using one of the AND NOT Boolean operators (**NOT**, **-**, **! &**), and entries that qualify based on one value and not another are selected.

For example, to find all the entries in a non-keyworded SI-path that begin with "NEW" except those that begin with the value "NEW YORK":

argument	= ˜**NEW@ NOT "NEW YORK";**	SQL Notation
argument	= ˜**NEW@-"NEW YORK";**	Infix Notation
argument	= **[NEW@][NEW YORK]! &**	Reverse Polish Notation

☞ **Incidentally, the same retrieval could also be performed more efficiently using indexed access:**

argument	= **NEW@<>"NEW YORK"**

Finding entries with combined Boolean operators

The Boolean operators used by SUPERDEX allows very powerful combinations of operations to be specified.

For example, to find all the entries in a keyworded SI-key that contain both the words "COMB" and "BIND" or "HOLD" and "DRILL" but not any word beginning with "FASTEN":

argument	= ˜**(COMB and BIND) or (HOLE and DRILL) not FASTEN@;**	
		SQL Notation
argument	= ˜**(COMB+BIND),(HOLE+DRILL)-FASTEN@;**	
		Infix Notation
argument	= **[COMB][BIND]&[HOLE][DRILL]& | [FASTEN@]! &**	
		Reverse Polish Notation

Processing of Boolean Operators for SQL Notation

At the lowest level, all processing of the arguments occur in Reverse Polish Notation. Therefore, the SQL Notation will be converted internally first to the Infix format and then to the RPN format.

Most of the examples and tables following will reference the SQL format. When the Infix format or RPN format is referenced, the example will be marked as such.

Since the SQL format is first converted to Infix, following are several examples of the conversion.

SQL Notation to Infix Notation Processing:	
SQL Argument	*Infix Argument*
-CHICAGO and JOHN;	CHICAGO+JOHN
-FITTING OR NIBBLE or CONNECTOR;	FITTING,NIBBLE,CONNECTOR
-NEW@ not "NEW YORK";	NEW@-"NEW YORK"
-A@ OR B@ OR C@;	A@,B@,C@

For information on how the Infix Notation is processed, see the following section.

Processing of Boolean Operators for Infix Notation

At the lowest level, all processing of the arguments occur in Reverse Polish Notation. Therefore, the Infix Notation will be converted internally to the RPN format.

Most of the examples and tables following will reference the SQL format. When the Infix format or RPN format is referenced, the example will be marked as such.

Following are several examples of the conversion.

Infix Notation to Reverse Polish Notation Processing:	
Infix Argument	*RPN Argument*
-CHICAGO+JOHN;	[CHICAGO][JOHN]&
-FITTING,NIBBLE,CONNECTOR;	[FITTING][NIBBLE]|[CONNECTOR]|
-NEW@-"NEW YORK";	[NEW@][NEW YORK]!&
-A@,B@,C@;	[A@][B@]|[C@]|

Active and backup SI-subsets

Besides the active SI-subset in which the SI-chain selected by an *argument* using one of the Boolean operators is copied, a *backup SI-subset* is used when processing *arguments* that contain more than one value.

The internal processing of the *argument* specified in the previous example is done as follows (RPN is used to define the process):

[COMB]   The SI-chain that is formed by the selection is stored in the active SI-subset

[BIND] &   A Boolean ANDing is performed between the result of this selection and the virtual SI-chain contained in the active SI-subset

[HOLE]   The virtual SI-chain contained in the active SI-subset is transferred to the backup SI-subset, and the resulting SI-chain is stored in the active SI-subset (replacing the existing SI-chain)

[DRILL] &   An ANDing is performed in the same manner as above

| The contents of the backup SI-subset and the active SI-subset are ORed, and the resulting SI-chain is stored in the active SI-subset. The SI-chain stored in the backup SI-subset is deleted.

[FASTEN@] ! & The negated result of the selection is ANDed with the SI-chain in the active SI-subset, with the resulting SI-chain stored in the active SI-subset, replacing the existing SI-chain

For consistency and efficiency, specify a Boolean operator following each bracketed value except the first (leftmost), as shown:

    [A@] [B@] | [C@] |

Alternately, specifying the *argument* value

    [A@] [B@] [C@] | |

will also select the desired entries but will destroy the backup SI-subset.

However, the *argument* value

    [A@] [B@] [C@] [D@] | | |

will not work.

The effects of the various operations on the active and backup SI-subsets is summarized in the following table. **All of the arguments are defined as RPN since this is the only way to directly manipulate the active and backup SI-subsets using all of the special arguments.**

argument	SI-chain in active SI-subset	SI-chain in backup SI-subset
&	ANDed with backup	deleted
|	ORed with backup	deleted
! &	AND NOTed with backup	deleted
[x]	replaced by x	replaced by old active
[x] &	ANDed with x	unchanged
[x] |	ORed with x	unchanged
[x] ! &	AND NOTed with x	unchanged
[x] $	ORed with active after x is ANDed with backup	unchanged
/	unchanged	replaced by active
[ ]	erased	replaced by active
\	replaced by backup	deleted
/\	unchanged	deleted
\/	replaced by backup	unchanged
^	swapped with backup	swapped with active
\[x]	replaced by x	unchanged
/[x] &	ANDed with x	replaced by old active
/[x] |	ORed with x	replaced by old active
/[x] ! &	AND NOTed with x	replaced by old active
![x] &	inverts and ANDs with x	unchanged
[*]	replaced by projection	replaced by old active

Successive refinement

The complex DBFIND *argument* illustrated previously may be broken up into several DBFIND calls rather than being performed in a single call.

For example, the SQL *arguments* would be:

1. DBFIND *argument* = ~ **(COMB and BIND);**
2. DBFIND *argument* = ~ **or (HOLE and DRILL);**
3. DBFIND *argument* = ~ **not FASTEN@;**

The result is the same as when using a single DBFIND call with the complete *argument* containing multiple values. After each DBFIND, the number of qualifying entries in the SI-chain (stored in the active SI-subset) is returned in the *status* array, and this may be reported to the user to decide at any stage whether or not to continue.

The **/** operator may be used to save an intermediate result in the backup SI-subset and retrieve it later using the **** operator.

Positioning on a virtual SI-chain

A similar technique to that described above may be used to position at the beginning or end of or at any entry on a virtual SI-chain in the active SI-subset.

DBFIND *modes* 1*nn* and 2*nn* may be used to position on the entry whose SI-key value matches the specified *argument*; if no matching entry exists, the internal SI-pointer is set to the location where the entry would reside. The following parameters could be used:

dset	= **CUSTOMER-MASTER**
mode	= **-103**
item	= **;** or **0**
argument	= **ABC**

In this example, the SI-pointer would be positioned at the SI-index for the customer ABC.

This same technique may also be used for going to the beginning of (rewinding) or end of a virtual SI-chain, using *modes* 100 and 200, respectively. The entries could then be retrieved with DBGET *modes* 15 and 16.

Determining entry count of a virtual SI-chain

In addition to being able to locate and retrieve entries on any virtual SI-chain in the active SI-subset, it is possible to determine the number of entries on any virtual SI-chain by calling DBFIND in *mode* 1 with a special *argument*, as shown:

base	= **CUST**
dset	= **CUSTOMER-MASTER**
mode	= **1**
item	= **;** or **0**
argument	= **@@**

This returns the entry count for the SI-chain that corresponds with the base and dataset specified in the *base* and *dset* parameters.

Finding entries using multiple SI-paths in a dataset

This same technique of using multiple DBFIND calls to refine a selection works not only on a single SI-path in a dataset but on multiple SI-paths.

For example, to find all customers that start with "GENERAL" and are located in "LOS ANGELES," two DBFIND calls would be performed and their results combined. The first DBFIND call would include these parameters:

item	= **CUSTOMER-NAME**
argument	= **¯GENERAL@;**

The second DBFIND call would specify a different SI-path and the *argument* that corresponds with that SI-path, as well as a Boolean operator indicating how the results should be combined:

```
item          = CITY
argument      = ˜and "LOS ANGELES";
```

The **AND** on the second DBFIND call indicates that the two sets should be ANDed. The second set can instead be logically ORed or AND NOTed, by specifying **OR** or **NOT** instead of **AND**.

For example, to find all the customers who have not placed any orders since January 1, 1991 OR have an average order amount of fifty dollars or less, two DBFIND calls are performed with the **OR** operator prefixing the *argument* on the second DBFIND:

```
item          = LAST-ORDER-DATE
argument      = ˜<=901231;

item          = AVG-ORDER-AMT
argument      = ˜or <=50;
```

Finding entries using multiple datasets

The same technique may be used to qualify entries across multiple datasets by using multiple DBFIND calls, each specifying a different dataset.

It is preferred that both datasets contain a common item that is used in an SI-subkey in each set. In this case, the common item forms a logical linkage between the two sets, and is referred to as the *SI-link*.

☞ **It is required that the item assigned as the SI-link be configured as an SI-subkey in the SI-path that the DBFIND is being called against; alternately, for SI-paths against a master dataset, it may be the IMAGE search master field.**

If there is no common item between the datasets defined as an SI-subkey, a *projection* may be performed, as will be explained later.

Let's look at an example that locates all the customers that are slow paying for orders and currently have unpaid orders. The customers are contained in the CUSTOMER-MASTER dataset which has CUSTOMER-NUMBER as its search field and a simple SI-path called *AVG-DAYS-TO-PAY*. The orders are contained in the ORDER-DETAIL set, which has an SI-path called *ORDER-STATUS* which is comprised of the SI-key items ORDER-STATUS and CUSTOMER-NUMBER.

Both datasets have the CUSTOMER-NUMBER in common, so this defines the SI-link used to logically join the two sets. The SI-link is declared as a second value in the *item* parameter on one or both DBFIND calls, in addition to the SI-path name, in this format:

```
item          = SI-path,SI-link
```

To accomplish the search, DBFIND is first called to locate all the unpaid orders with the specified SI-path and SI-link:

```
dset        = ORDER-DETAIL
item        = ORDER-STATUS,CUSTOMER-NUMBER
argument    = ˜UNPAID;
```

Then, DBFIND is called again to locate all the customers that take an average of more than 45 days to pay. An **AND** prefixes the *argument* value to cause the virtual sets to be ANDed:

```
dset        = CUSTOMER-MASTER
item        = AVG-DAYS-TO-PAY,CUSTOMER-NUMBER
argument    = ˜and >=45;
```

Alternately, the virtual sets could have been ORed or AND NOTed by substituting the **OR** or **NOT** operator in place of the **AND** operator.

After both DBFIND calls are completed, DBGETs could be performed against CUSTOMER-MASTER to retrieve the qualifying entries.

In this example, there are two virtual SI-chains in the active SI-subset. For this reason, the AND is done on the values of the SI-link. If instead there were only one SI-chain in the active SI-subset, the AND would be done on the SI-extension (the search field value for masters or relative record number for details; refer to the *Internal structures* appendix for more information about the layout of the SI-subset).

Finding corresponding entries in multiple datasets

When DBFIND is called in succession against multiple datasets, one SI-chain per dataset is placed into the active SI-subset. Entries may be retrieved from any of these virtual SI-chains, independent of one another, simply by using DBGET with the appropriate dataset specified in the *dset* parameter.

Since the entries on these SI-chains are logically related by the SI-link, it is often desirable to find entries on one or more of the SI-chains whose SI-link values match a specified value, thereby performing a search against a virtual SI-chain in the active SI-subset rather than against entries in a dataset.

This technique is facilitated by performing a DBFIND on each SI-chain with the exact SI-link value specified in the *argument* parameter and **;** or **0** specified in the *item* parameter. **It is the "null item" that causes the DBFIND to act on the active SI-subset rather than the dataset.** The exact SI-link value must be specified.

For example, to find all the customers whose SI-link value is equal to "ACME" call DBFIND with the following:

```
dset        = CUSTOMER-MASTER
mode        = 1
item        = ; or 0
argument    = ACME
```

Subsequent DBGETs in *mode* 5 or 6 will access the sub-selected entries that have an SI-link value of "ACME," while DBGET *modes* 15 and 16 will access the entire virtual SI-chain.

Finding entries using multiple databases

Entries in multiple databases may be located in very much the same way as those in multiple datasets. Again, multiple DBFIND calls are used with an SI-link, but each has a different value for the *base* parameter. If a different item in each base is used as the SI-link, they must be configured with the same length.

SUPERDEX requires that DBFINDs against multiple databases be logically linked together. If both DBFIND calls are performed in immediate succession (with no intermediate intrinsic calls) and therefore use the same *status* array, SUPERDEX automatically links the bases together, and no specification is required by the program.

If intermediate calls are performed, the program must logically link the bases. To facilitate this, DBFIND returns a unique number in the second word of the *status* array (unused by IMAGE). The program must retrieve this number from the *status* array of the first DBFIND and specify it in the second word of the *status* array in the second DBFIND.

Let's look at an example of two databases, one containing customers and the other sales history. We want to determine sales trends of books to schools in the CUST base by reviewing historical data in the SALES base.

The relevant sets in each base have the common item CUSTOMER-NUMBER, so this item will be used as the SI-link. If no common item exists, a *projection* may be done, as we will see shortly.

The first DBFIND call locates the customers that are schools using a keyworded SI-key. The *argument* values are partial keys being ORed together:

```
base      = CUST
dset      = CUSTOMER-MASTER
item      = CUSTOMER-ID,CUSTOMER-NUMBER
argument  = ˜SCHOOL@ or UNIVERSITY@ or COLLEGE@;
```

The second DBFIND call accesses the appropriate dataset in the other database:

```
base      = SALES
dset      = PART-SUMMARY
item      = VENDOR-ID,CUSTOMER-NUMBER
argument  = ˜and (MCMIL@ or MCGRAW@);
```

☞ Both calls specify the SI-link as the second value in the *item* parameter. Also, the use of **AND** at the beginning of the *argument* in the second DBFIND call tells SUPERDEX to AND its results with the previous DBFIND call.

As always, the **OR** or **NOT** operator could have been used instead of the **AND** operator to perform an OR or AND NOT retrieval between databases.

Finding entries in multiple sets and bases using projection

In the last two examples, a common item exists between the two datasets and databases being searched, and was defined as the SI-link. For situations in which there is no common item but a logical relationship exists, *projection* may be used.

A projection is an operation that permits two datasets that do not contain a common item to be linked together, providing each has an item in common with a third dataset. For this discussion, we'll refer to the first dataset as set *A*, the second dataset as set *B*, and the third (linking) dataset as set *C*. The projection reassigns the SI-link from the item set *C* has in common with set *A* to its item in common with set *B*, thereby forming a logical relationship between set *A* and set *B*, even though they do not contain a common item.

A projection is invoked by a separate DBFIND against set *C* which is called between the DBFINDs against set *A* and set *B*. Internally, a projection takes the SI-link values returned internally by the DBFIND against set *A* (which are stored in the active SI-subset), looks up the corresponding entries in set *C*, and replaces them with the SI-link values that will be used for set *B*. The SI-link for set *A* and set *B* are both defined by item name or number in the *item* parameter, as shown:

        item   = SI-path,new SI-link

where *SI-Path* is a concatenated SI-Path that is comprised of the *old SI-link* as SI-Subkey-1 and the *new SI-link* as any other SI-Subkey.

Let's look at an example that locates all quotations for earthquake coverage given to policyholders in Los Angeles in November and December of 1987. This requires four DBFIND calls against three datasets, with the second DBFIND call performing the projection.

The first DBFIND locates all policyholders in Los Angeles, with SI-link specified in the *item* parameter:

        dset   = INSURED-MASTER
        item   = CITY,POLICY-NUMBER
        argument   = ˜"LOS ANGELES";

The next DBFIND performs the projection, as designated in the *argument* by the special ∗ (asterisk) operator. The *item* specifies the SI-link used in the previous DBFIND, as well as the SI-link that will be used in the next DBFIND:

        dset   = QUOTE-MASTER
        item   = POLICY-NUMBER,QUOTE-NUMBER
        argument   = [∗]

The third DBFIND accesses the same dataset in which the projection was performed and locates all the entries in the specified date range. The new SI-link is specified in the *item* parameter:

        dset   = QUOTE-MASTER
        item   = QUOTE-DATE,QUOTE-NUMBER
        argument   = ˜and >=871101<=871231;

The final DBFIND locates all the entries in another dataset that are of the requested coverage type:

dset	= **QUOTE-DETAIL**
item	= **COVERAGE-TYPE,QUOTE-NUMBER**
argument	= **¯and EQ;**

Notice that the last two DBFIND calls used **AND**s in the *argument* to logically AND the results.

☞ **Projection uses all qualifying SI-link values contained in the active SI-subset in locating entries. If there is more than one SI-chain in the active SI-subset, entries may qualify more than once. Therefore, a projection should only be performed when there is only one SI-chain in the active SI-subset.**

Circumstances in which the SI-link must be specified

As described, the SI-link is a common item that is configured in an SI-subkey in each set and which is used to form a logical linkage between the two sets. It is required that the item assigned as the SI-link be configured as an SI-subkey in a concatenated SI-key; alternately, for SI-paths against a master dataset, it may be the IMAGE search master field.

The SI-link need not be specified in all cases involving relational access against multiple datasets and/or databases, and may be omitted on some DBFIND calls. The following rules govern the specification of the SI-link:

1. For relational access between two different SI-paths within the same dataset, the SI-link is not required. The Boolean operations are based only on the SI-extension (the search field value for a master or relative record number for a detail).

2. For relational access between two datasets in the same database, the SI-link must be specified for the first DBFIND call but not the second, although there is no harm in specifying it for both DBFIND calls. In this case, the value of the SI-link is used for comparison instead of the SI-extension. The SI-link need not be specified for a manual master for which the IMAGE search field is the default SI-link.

3. For relational access between databases, the SI-link is always used for comparison and must always be specified for every DBFIND call. (Additionally, if intermediate calls between databases are performed, the bases must also be logically related via word 2 of the *status* array, as described previously.)

Qualifying entries in the active SI-Subset

While SI-Subsets were designated for relational retrievals, they can be very useful for ultra fast access since they can be considered as in-memory datasets.

An in-memory dataset is created by a DBFIND call, where the SI-Link is specified and the argument is a relational argument. This SI-Subset acts like a dataset with SI-Link as a SUPERDEX key. You can access this set by DBFIND utilizing most of the SUPERDEX features.

For example, first create an SI-Subset for customer "**ACME**".

dataset	=	**CUSTOMER-MASTER**
mode	=	**1**
item	=	**CUSTOMER-NAME,ORDER-NUMBER**
argument	=	**~ACME;**

Now do a selection on this SI-Subset with order-number **70123**:

dataset	=	**CUSTOMER-MASTER**
mode	=	**1**
item	=	**; or 0;**
argument	=	**70123**

Instead of using mode 1, the special modes 1*nn* and 2*nn* can be used to retrieve partial keys in sorted order. **Partial key retrieval using the @ operator and the ? or # match characters are NOT available.**

☞ **In-memory datasets can be especially useful for applications where you expect many DBFINDs to fail, or when you need the same information frequently, since you get this information without any disc access.**

Preparing the argument

The many *argument* operators available with SUPERDEX allow very advanced and powerful retrievals using the simple DBFIND *mode* 1. *Arguments* can contain one or more of the following in various combinations:

- the @, << >>, ?, and # conditional operators
- the <=, >=, and <> relational operators
- the **AND, OR**, and **NOT** Boolean operators
- the +, ,, and − Boolean operators
- the &, |, and !& Boolean operators
- the *, /, \, $, and ^ special operators

Allowing these generalized *arguments* for DBFIND *modes* 1 and 10 permits standard lookup routines to perform many types of retrievals, with the *argument* determining the type and scope of access. Complex *arguments* may be prepared for DBFIND calls by several methods.

The simplest method, which requires little or no reprogramming, is to have users specify the entire *argument* themselves, including multiple values and various operators, and have the program pass them literally to DBFIND.

Another method is to assign function keys to facilitate various retrieval capabilities. The user could enter a string and hit a function key, and the program could read the function key label and concatenate the appropriate operator to the specified value to form the *argument*.

Various other methods are available for programmatically constructing the DBFIND *argument*, such as prompting the user with various selection boxes to check off or enter values into.

Effect of DBFIND on the SI-pointer and current path

Like IMAGE, SUPERDEX returns a condition word of zero for successful DBFIND calls and non-zero if an error or exception is detected.

In IMAGE, calling DBFIND against a master set returns condition word -21 ("**SPECIFIED INTRINSIC IS NOT ALLOWED ON MASTER SET**"); in SUPERDEX, no error is returned because it is a valid operation.

☞ **Since the condition word -21 is not returned on the master dataset for a DBFIND, some generic access programs may need modification. If a DBFIND is used to determine whether a dataset is a master or detail, this will not work. If a master dataset has at least one SI-Path, the DBFIND will not return the -21, but can return the condition word -52 (see following).**

If the specified *item* is neither a valid SI-path nor IMAGE path, condition word -52 ("**ITEM SPECIFIED IS NOT AN ACCESSIBLE SEARCH ITEM IN THE SPECIFIED SET**") is returned.

If DBFIND does not find an entry that matches the specified *argument*, condition word 17 ("**NO ENTRY FOUND**") is returned. If the SI-path name is the same as the IMAGE search field name and the SUPERDEX DBFIND against the SI-path fails, an IMAGE DBFIND is automatically performed against the search field. If called in *mode* 1 or 10, the SI-pointer is not set and the current path is reset to the dataset's current IMAGE path. Therefore, if the condition word is ignored and subsequent DBGETs are called, they will operate on an IMAGE path (the current IMAGE path for a detail set or the synonym chain for a master set) rather than the SI-path.

If DBFIND is called with *mode* 1*nn* or 2*nn*, the SI-pointer is set immediately before or after the nearest qualifying entry and the current path is set to the appropriate SI-path and does not change. For example, DBFIND *mode* 102 with an *argument* of **BRAC** may not locate a matching entry but will set the SI-pointer before "BRADMARK," the nearest qualifying entry. Then, DBGET *mode* 15 or 16 may be used to retrieve the entries in ascending or descending order.

The following table summarizes the effects of DBFIND on the SI-pointer and current SI-path:

mode	condition word = 0	condition word <> 0
1	before entry	current path not set ***
10	before entry	current path not set **
100*	before first (alphabetical) entry	current path not set
1nn	before entry	if cw = 17, before next entry
200	after last (alphabetical) entry	current path not set
2nn	after entry	if cw = 17, after next entry

*   may be followed by either DBGET *mode* 5 or 15 or *mode* 6 or 16 and will start at either the beginning or end of the dataset, respectively, as compatible with IMAGE

**   if using relational access, the current path is retained

*** if using relational access, the current path is retained; otherwise, defaults to current IMAGE path

Retrieving entries with DBGET

Entries that are located with DBFIND may be returned in ascending sorted sequential order with *modes* 5 and 15 and descending order with *modes* 6 and 16.

Like IMAGE, *modes* 5 and 6 return condition words 14 and 15 (**"BEGINNING OF CHAIN"** and **"END OF CHAIN"**) when all qualifying entries on the SI-chain have been returned. *Modes* 15 and 16 continue to return entries in sorted sequential order that are not part of the SI-chain, like greater-than-or-equal-to and less-than-or-equal-to retrievals.

DBGETs with Un-initialized SI-chain

If the SI-chain has not been established or the SI-pointer is outside of the current SI-chain, this could be the result of:

- DBFIND with *mode* other than 1 or 10
- DBFIND that does not find a match (returns condition word 17)
- DBGET *mode* 4

In these cases, DBGET *modes* 15 and 16 should be used instead of *modes* 5 and 6; otherwise, the results are unpredictable.

Repositioning on an SI-chain

If reading an SI-chain (not an SI-subset) along an SI-path that has a *unique* relationship (one SI-index per data record, unlike with a keyworded SI-path), DBGET *mode* 4 can be used to reposition on the SI-chain. An application for this is, for example, implementing a "previous page" function when displaying entries on a terminal. This can be programmed by keeping an internal list of relative record numbers of the first entry on each page and then returning to any page by calling DBGET *mode* 4 followed by DBGET *mode* 5s in a loop.

To reposition on an SI-chain after switching to a different SI-path or IMAGE path in the same dataset, save the relative record number before switching paths, then to return, call DBFIND *mode* 10 with the original search argument followed by DBGET *mode* 4 using the saved record number. From here, DBGET *modes* 5, 6, 15, and 16 may be performed normally.

Reading SI-indices only

It is desirable for efficiency to restrict the DBGETs to reading only the SI-indices rather than actually retrieving the entries from the datasets whenever possible.

When performing indexed access, the full SI-index including the SI-extension (search field value for a master dataset or relative record number for a detail) is returned. For relational access, the SI-link (if specified) and SI-extension are returned.

Several operations that may be accomplished by reading SI-indices:

- validating the format of any value in an SI-index, which always includes the search field value for SI-paths related to master datasets.
- intermediate storage of the SI-extension to facilitate resetting the SI-pointer to its former position, such as after changing SI-paths
- testing for the presence or absence of a value in multiple SI-paths by reading them in parallel

To read SI-indices only, use a list of **!** with DBGET *modes* 5, 6, 15, or 16.

To retrieve the entry associated with any SI-index, use DBGET *mode* 1 (reread current entry).

☞ The **!** *list* is never transferred to IMAGE; therefore, the *list* in use before the **!** was declared will still be active.

Reading multiple SI-indices with a single DBGET

When using the **!** *list* to read SI-indices only, it is possible to read multiple SI-indices with a single DBGET, equivalent to calling DBGET multiple times in a loop.

This is facilitated by an optional *SI-counter* parameter, which specifies the number of SI-indices to return. If not specified, the default SI-counter value is 1. The SI-counter is a numeric literal immediately following the **!**, as shown:

> *list* = **!** *SI-counter*
> *list* = **! 5 0**

If a beginning-of-chain or end-of-chain condition is detected during the DBGET call, the returned indices are less than the requested number of indices. The number of indices returned is contained in word 5-6 of the status array. The SI-Pointer is positioned at the first/last index.

☞ **If using an SI-counter, be sure to use a *buffer* large enough to accommodate all the SI-indices that will be returned.**

☞ The SI-counter parameter cannot be used with any *list* construct other than **!** .

DBGET used to locate entry for DBUPDATE

If the DBGET used to locate an entry for updating specifies neither the **@** *list* nor a *list* that includes all the SI-subkeys whose values are being changed, SUPERDEX will automatically perform one or more rereads (DBGET *mode* 1) with various *lists* when performing the DBUPDATE.

This is both inefficient and causes the current *list* to change without the knowledge of the program, and therefore any subsequent calls done without re-initializing the *list* may be faulty. It is therefore recommended that the **@** *list* or a *list* that contains all SI-subkeys be used when calling DBGET before DBUPDATE or when calling DBUPDATE.

DBGET used to locate entry for DBDELETE

Similarly, if all SI-keys are not included in the *list* of the DBGET used to locate an entry for deletion, SUPERDEX will automatically change the *list* to @ (unless it is already @ or *;*).

Because the current *list* may change and therefore any subsequent calls done without re-initializing the *list* may be faulty, it is recommended that the @ *list* or a *list* containing all SI-keys be used when calling DBGET before DBDELETE (the *;* *list* may also be used but is inefficient).

Effect of DBGET on the SI-pointer and current path

Like IMAGE, SUPERDEX returns a condition word of zero in the *status* array for successful DBGET calls and non-zero if an error is detected.

In addition, DBGET *modes* 5 and 6 performed on a master set read down and up the SI-chain, whereas in IMAGE, they read down and up the current synonym chain.

If the *item* specified is neither a valid SI-path nor IMAGE path, condition word -52 ("**ITEM SPECIFIED IS NOT AN ACCESSIBLE SEARCH ITEM IN THE SPECIFIED SET**") is returned.

The following table summarizes the effects of DBGET on the SI-pointer and current SI-path:

mode/relationship	condition word = 0	condition word <> 0
5 or 15*	before next entry	if *cw* = 15, no change
6 or 16*	after previous entry	if *cw* = 14, no change
other unique	on entry	no change
other non-unique	no change	no change

* if switching from *mode* 5 or 15 to *mode* 6 or 16, the same entry is not returned twice.

Serially Reading All Entries

Like IMAGE, SUPERDEX will allow a serial read through a dataset using DBGET mode 2. This will return the records in the same order as IMAGE.

If you wish to sequentially access all the records in sorted order by any path, initialize the path using DBFIND mode 100, and retrieve them with DBGET mode 15. This will return the records in sorted order, based on the SI-Path defined, and return a condition word of 11 (End-of-file), as DBGET mode 2 does.

For master datasets, using DBFIND mode 100 and DBGET mode 15 can be faster and more efficient than DBGET mode 2. SUPERDEX does not need to read each block in the dataset, while IMAGE needs to read each block to see if any records exist in the block.

Additional programming considerations

Summary of effects of SI-intrinsics on the SI-pointer and current SI-path

The table below indicates the effects of various intrinsics on the SI-pointer and current SI-path.

In performing a DBPUT, DBPUTIX, DBDELETE, DBDELIX, or DBUPDATE against an SI-path for which more than one SI-index may point to the same record (a *n*-to-one relationship referred to below as a *non-unique* relationship as in keywording, grouping, compound items, and SIUSER), the position of the SI-pointer does not change--regardless of whether or not a unique relationship exists for a given entry.

intrinsic/mode	condition word = 0	condition word <> 0
DBFIND mode 1	before entry	current path not set ***
DBFIND mode 10	before entry	current path not set ***
DBFIND mode 100*	before first entry	current path not set
DBFIND mode 1*nn*	before entry	if *cw* = 17, before next entry
DBFIND mode 200	after last entry	current path not set
DBFIND mode 2*nn*	after entry	if *cw* = 17, after next entry
DBGET modes 5/15**	before next entry	if *cw* = 15, no change
DBGET modes 6/16**	after previous entry	if *cw* = 14, no change
DBGET other modes unique	on entry	no change
DBGET other modes non-unique	no change	no change
DBPUT	no change	no change
DBPUTIX	no change	no change
DBUPDATE	no change	no change
DBDELETE	no change	no change
DBDELIX	no change	no change

*   may be followed by either DBGET *mode* 5 or 15 or *mode* 6 or 16 and will start at either the beginning or end of the dataset, respectively, as compatible with IMAGE

**   if switching from *mode* 5 or 15 to *mode* 6 or 16, the same entry is not returned twice

***   if using relational access, the current path is retained

Testing for the existence of SUPERDEX;

New DBINFO *mode* 311 is provided to test for the presence of SUPERDEX, and returns information about the SI-paths configured for a specified dataset. It is designed for programs that are run against the same database on various systems, of which some do not contain SI-paths.

Additionally, new DBINFO *mode* 312 may be used to determine the characteristics of a specific SI-path.

Calling SUPERDEX intrinsics in Privileged Mode

☎ All SUPERDEX intrinsics should be called in user mode. If called in Privileged Mode (after a call to GETPRIVMODE), contact Bradmark's Technical Support for information on possible problems.

PREParing programs

Programs that access SUPERDEX'ed databases must be :PREPped with DS and optionally, MR capability; DS capability is not required for Native Mode programs under MPE/XL. Programmers who :PREP programs that use SUPERDEX will need one or both of these capabilities added to their users with :ALTUSER. Users running the programs do not require additional capabilities.

All SUPERDEX intrinsics automatically extend the stack, by:

3 * *SI-dataset block size*

An additional 1 or 2 Kwords of stack space is required if performing relational access.

Programs should be :PREPped with a larger *MAXDATA* parameter value of at least 2 - 3 times the block size (in words) of the SI-dataset. Some programs will need to be executed with *;NOCB*.

BASIC/3000

☎ If using BASIC/3000, please contact Bradmark's Technical Support for instructions.

BRW (Business Report Writer)

SUPERDEX is capable of interfacing to HP's Business Report Writer package. The Full PowerHouse option is required for the BRW interface. Please see the *Business Report Write interface* in the *Fourth-Generation Language User Manual.*

Business Basic

If using SUPERDEX with Business Basic, make sure the necessary SL segments have been properly copied, as explained in the Installation chapter of the Configuration / Establishing SI-indices section. Please see the *Business Basic Interface* supplement.

C

No special considerations exist for programs written in C.

COBOL

COBOL programs that call SORT using **INPUT PROCEDURE** and **OUTPUT PROCEDURE** require a large amount of stack space, and calling an SI-intrinsic from **INPUT** or **OUTPUT PROCEDURE** may cause a stack overflow. If this happens, limit the amount of stack used for sorting in COBOL programs by including the following command in source programs and then recompiling:

$CONTROL SORTSPACE = *nnnnn*

where *nnnnn* is the number of words allocated for sorting, between 0 and 24000.

If COBOL programs are written in COBOL or COBOLII that does not support $CONTROL SORTSPACE yet, then use the USING statement.

FORTRAN

No special considerations exist for programs written in FORTRAN, although SORTINIT may be used to restrict stack usage for sorting if necessary.

Pascal

No special considerations exist for programs written in Pascal, although SORTINIT may be used to restrict stack usage for sorting if necessary.

PowerHouse

☎ SUPERDEX has interfaces to Cognos' PowerHouse modules QUICK, QUIZ, and QTP available, with separate software and documentation. Contact your Bradmark sales representative or distributor for information and a demonstration.

RPG

If the IMAGE intrinsics are being coded and called directly by the programmer, SUPERDEX does not require an interface. If the intrinsics are not handled in this manner, SUPERDEX can not be used with RPG.

SPL

No special considerations exist for programs written in SPL, although SORTINIT may be used to restrict stack usage for sorting if necessary.

TRANSACT

Programs written in TRANSACT are converted quite easily to use SUPERDEX. Basically, the SI-path name is placed into the *key* register, the search argument into the *argument* register, and **FIND(CHAIN)**, **FIND(RCHAIN)**, or **PATH** is called against a master or detail dataset. Please see the *TRANSACT Interface* supplement for more information and documentation.

PROTOS

SUPERDEX does not require an interface with PROTOS.

Native Language Support

Adding, updating, and indexing entries

When Native Language Support (NLS) is activated for a database, SI-indices are generated by SIMAINT, DBPUT, DBPUTIX, DBUPDATE and SI-USER according to HP's documented NLS collating sequences. This may result in some confusion because indexing may be done differently with and without NLS.

The following table shows how SI-indices for a concatenated SI-key would be sorted with and without NLS:

with NLS		without NLS		Keywords without NLS	
PAPER	100	PAPER	100	PAPER	100
PAPER	400	PAPER	400	PAPER	200
Paper	300	PAPERBAG	150	PAPER	300
paper	200	Paper	300	PAPER	400
PAPERBAG	150	paper	200	PAPERBAG	150

Qualifying entries with DBFIND

To qualify the entries with the SI-keys shown above, the *arguments* **PAPER@**, **Paper@**, and **paper@** are treated equivalently and all of the above entries are returned in the order shown in the left column.

☞ **Since the different representations of the same word are stored as distinct indices, there are circumstances where the same entry may qualify more than once.**

Overview

This section describes the various SI-intrinsics provided with SUPERDEX as enhancements to the IMAGE intrinsics as well as the new DBERASE, DBPUTIX, DBDELIX intrinsics and SIUSER procedure.

Chapter 1	**Enhancements**
Description	Briefly documents the _Enhancements_ to the SI-intrinsics over their IMAGE counterparts.
	Each subsequent chapter documents one or more intrinsics, including a discussion of each, their syntax (if different than IMAGE), and parameters (if different than IMAGE). The intrinsics are listed in alphabetical order by name.

Enhancements

For every IMAGE intrinsic there is an equivalent SUPERDEX intrinsic that has the same name and uses the same parameters. Some additional parameter values are available to achieve new functionality.

The enhancements to standard IMAGE intrinsics are summarized alphabetically by intrinsic in the table on the following page.

Because most SI-intrinsics are functionally and syntactically identical to their IMAGE counterparts, the changes are transparent; therefore, generally only variations are documented here.

Summary of Intrinsic

DBBEGIN	no changes; automatically imposed if missing on DBPUT, DBUPDATE, DBDELETE, DBPUTIX, and DBDELIX and base enabled for logging. MDBX not supported.
DBCLOSE	no changes
DBDELETE	automatically maintains SI-indices
DBDELIX	new intrinsic; explicitly deletes SI-index
DBEND	no changes; automatically imposed if missing on DBPUT, DBUPDATE, DBDELETE, DBPUTIX and DBDELIX and the base is enabled for logging. MDBX not supported.
DBERASE	new intrinsic; erases dataset and associated SI-indices in a fast mode
DBERROR	no changes, no enhancement intrinsic provided
DBEXPLAIN	no changes, no enhancement intrinsic provided
DBFIND	- works the same on master and detail dataset - new *mode* 10 works same as *mode* 1 but does not return qualifying entry count - new *modes* 100-299 position SI-pointer - *argument* may contain multiple values and operators - multiple calls perform dynamic queries on multiple fields, sets, and bases - can qualify a master entry based on its related detail entries in a super-group - the results of the current and previous DBFIND are maintained
DBGET	- works the same on master and detail datasets - *modes* 5 and 6 retrieve entries in ascending and descending sorted order - new *modes* 15 and 16 return all entries in the dataset alphabetically - new *!list* reads SI-indices only; optional *SI-counter* returns multiple SI-indices
DBINFO	new modes 311 and 312 return information about SUPERDEX configuration
DBLOCK	no changes; conditionally imposed for DBPUT, DBUPDATE, DBDELETE, DBPUTIX, and DBDELIX
DBMEMO	ignored if logging is not enabled
DBOPEN	- internally establishes SUPERDEX configuration - replaces first word of IMAGE base name with the SDU number, instead of the DBU number

DBPUT	- automatically maintains SI-indices
	- entries with blank SI-keys are by default not indexed
DBPUTIX	new intrinsic; explicitly adds SI-index
DBUNLOCK	no changes; conditionally imposed for DBPUT, DBUPDATE, DBDELETE, DBPUTIX, and DBDELIX
DBUPDATE	- automatically maintains SI-indices
	- may be used to update SI-keys
DBXBEGIN	not supported
DBXEND	not supported
DBXUNDO	not supported
SITRANSLATE	new procedure to convert an *argument* from Infix Notation to Reverse Polish Notation
SIUSER	new user-written procedure; permits customer-defined SI-indices

DBBEGIN intrinsic

There are no syntax changes to the DBBEGIN intrinsic.

If the base is enabled for logging, SUPERDEX automatically imposes DBBEGINs (and DBENDs) around several intrinsics, if a DBBEGIN is not already in effect. Those intrinsics are:

- DBPUT
- DBPUTIX
- DBDELETE
- DBDELIX
- DBUPDATE

The DBEND is imposed regardless of whether or not the intrinsic call is successful.

If the DBPUTIX and DBDELIX intrinsics are used in combination with DBPUT and DBDELETE, the program should explicitly call DBBEGIN before the DBPUT or DBDELETE, and DBEND after the DBPUTIX or DBDELIX.

☞ Multiple database transactions (MDBX) is not supported at this time.

DBCLOSE intrinsic

The DBCLOSE intrinsic is the same as in IMAGE, although it additionally releases the run-time SUPERDEX structures.

☞ SUPERDEX may be configured to perform two DBOPENs against a database for greater efficiency. If this is the case, two DBCLOSEs are also performed automatically.

If the SI-indices are kept in a separate SI-index base, SUPERDEX automatically DBOPENs and DBCLOSEs the SI-index base.

DBDELETE intrinsic

Functionally the same as IMAGE, with the addition that corresponding SI-indices are automatically deleted, including super-grouped SI-indices and custom-defined SI-indices generated by the SIUSER procedure.

Locking

A DBDELETE against a SUPERDEX'ed set may also internally access the corresponding SI-dataset to delete related SI-indices. This should be taken into consideration when locking, since the appropriate SI-dataset must be locked during a DBDELETE, which may be done implicitly or explicitly. Refer to the _Locking_ chapter of the _Programming_ section for a discussion of the various locking methods.

Logical transactions

If the base is enabled for logging, SUPERDEX automatically imposes DBBEGINs and DBENDs around each DBDELETE that does not already specify them. The DBEND is imposed regardless of whether or not the DBDELETE is successful.

DBGET used to locate entry for DBDELETE

If all SI-keys are not included in the _list_ of the DBGET used to locate an entry for deletion, SUPERDEX will automatically change the _list_ to @ (unless it is already @ or ≀).

Because the current _list_ may change and therefore any subsequent calls done without re-initializing the _list_ may be faulty, it is recommended that the @ _list_ or a _list_ containing all SI-keys be used when calling DBGET before DBDELETE (the ≀ _list_ may also be used but is inefficient), unless the database has been opened by SUPERDEX a second time (see Section 3).

DBDELIX intrinsic

DBDELIX is a SUPERDEX intrinsic used to explicitly delete SI-indices from B-trees (its counterpart is the SUPERDEX DBPUTIX intrinsic, which explicitly adds SI-indices).

The DBDELIX intrinsic accesses only the appropriate SI-dataset and is used to maintain independent indices. It also provides a method for deleting custom SI-indices in addition to those removed automatically by DBDELETE.

Refer to the *Adding, updating, and deleting entries* chapter of the *Programming* section for further discussion.

Syntax

DBDELIX *(base,dset,mode,status,item,buffer)*

The DBDELIX intrinsic is syntactically similar to DBDELETE except that the *list* parameter is replaced by an *item* parameter and the *buffer* parameter contains the full SI-index, including the SI-extension.

Parameters

Base   The *base-ID* (same as IMAGE).

Dset   Name or number of the dataset in which the corresponding data entry exists.

             If accessing an independent SI-path, this parameter should be left blank or set to 200.

Mode   An integer with the value 1.

Status   Only the condition word is set.

Item   The name of the SI-path from whose B-tree to delete the specified SI-index.

Buffer   The full SI-index value including the extension. The extension may be:

             ■ the search field value for a master dataset
             ■ the relative record number for a detail dataset
             ■ a suitable user-defined value for independent SI-paths

Locking

A DBDELIX against an SI-path internally accesses the corresponding SI-dataset to delete one or more SI-indices. This should be taken into consideration when locking, since the appropriate SI-dataset must be locked during a DBDELIX, which may be done implicitly or explicitly. Refer to the *Locking* chapter of the *Programming* section for a discussion of the various locking methods.

Logical transactions

If the base is enabled for logging, SUPERDEX automatically imposes DBBEGINs and DBENDs around each DBDELIX that does not already specify them. The DBEND is imposed regardless of whether or not the DBDELIX is successful. If DBDELIX is used in combination with DBDELETE, the program should include a DBBEGIN before the DBDELETE and the DBEND after the DBDELIX.

Error handling

Since SUPERDEX uses standard IMAGE messages to report all errors and exceptional conditions, the same messages that are used by DBDELETE are displayed for DBDELIX.

DBEND intrinsic

There are no syntax changes to the DBEND intrinsic.

If the base is enabled for logging, SUPERDEX automatically imposes DBENDs (and DBBEGINs) around several intrinsics if a DBBEGIN is not already in effect. Those intrinsics are:

- DBPUT
- DBPUTIX
- DBDELETE
- DBDELIX
- DBUPDATE

The DBEND is imposed regardless of whether or not the intrinsic call is successful.

If the DBPUTIX and DBDELIX intrinsics are used in combination with DBPUT and DBDELETE, the program should explicitly call DBBEGIN before the DBPUT or DBDELETE, and DBEND after the DBPUTIX or DBDELIX.

☞ **Multiple Database Transactions (MDBX) are not supported at this time.**

DBERASE intrinsic

DBERASE is a SUPERDEX intrinsic that erases the contents of a dataset and also removes all corresponding SI-indices for all associated SI-paths. It is considerably faster than DBDELETEing all entries.

All entries in the dataset are read serially and erased, then the corresponding SI-indices are erased. As with DBDELETE, a master dataset cannot be erased if any of its entries have related detail entries--if they do, the erase will not be performed at all.

Syntax   **DBERASE** *(base,dset,mode,status)*

Parameters

Base   The *base-ID* (returned by DBOPEN).

Dset   The name or number of the master or detail dataset to erase.

Mode   An integer with the value 1.

Status   Standard IMAGE *status* array.   Only the condition word is set (to zero if successful or to one of the IMAGE condition words returned by DBDELETE if unsuccessful).

Improved speed in exclusive access mode

Although erasing a dataset with DBERASE is much faster than using DBDELETEs, its speed can be increased further if run in exclusive access mode by logging on as the database creator and executing the following DBUTIL commands:

```
:RUN DBUTIL.PUB.SYS

>>DISABLE base FOR ILR

  ILR is disabled.
>>DISABLE base FOR LOGGING

  Logging is disabled.
>>ENABLE base FOR AUTODEFER

  Autodefer is enabled.
>>EXIT

END OF PROGRAM
```

Don't forget to reset these run-time options back to their original status using the ENABLE command after the DBERASE has completed.

Recovery after abnormal abort

If the DBERASE fails due to a program abort, system failure, or other interruption, erase the dataset using a utility (such as DBGENERAL) or delete any remaining entries using QUERY.PUB.SYS and use the SIMAINT utility to reorganize all SI-paths related to the dataset. If SIMAINT is unable to successfully recover the database, use the SIMAINT with the DBLOAD entry-point.

DBERROR and DBEXPLAIN intrinsics

There are no changes to the DBERROR or DBEXPLAIN intrinsics, and no enhancement intrinsics are required.

The error conditions reported by DBERROR and DBEXPLAIN are standard IMAGE error messages, but may reflect errors encountered not only on the dataset containing the entries but also against an SI-dataset.

When analyzing an error, keep in mind that it may be related to an SI-dataset rather than (or in addition to) the dataset containing the entries, even though the entry dataset may be reported as being in error. The same is true when SUPERDEX is configured to maintain its SI-indices in a separate SI-index base: an error on the SI-index base will be reported against the primary base containing the data entries.

Refer to the table of *SUPERDEX intrinsic error and exceptional conditions* in the *Error and exceptional conditions* appendix for different types of errors, their meanings, and recommended actions.

Detecting structural inconsistency

If there is an inconsistency between a B-tree and its corresponding dataset such that an SI-index refers to a non-existent entry, condition word 17 ("no ENTRY FOUND") is returned. This would be caused by deleting an entry but not its related SI-index. In this case, the SI-path should be reorganized using the SIMAINT utility.

If an internal inconsistency within a B-tree is detected (due to program abort, system failure, etc.), condition word 18 ("BROKEN CHAIN") is returned. In this case, reorganize all SI-paths using the SIMAINT with the DBLOAD entry-point.

Refer to the *Maintenance and utilities* section for related information.

DBFIND intrinsic

DBFIND accesses an SI-path and sets the SI-pointer within that SI-path's B-tree for subsequent DBGETs. In addition to IMAGE, DBFIND works on both master and detail sets.

In using DBFIND against SI-paths, the *argument* may contain partial keys as well as generic values, relational operations, ranges, and multiple values which are logically combined via boolean operators. For keyworded SI-paths, the *argument* may contain a keyword; for concatenated SI-keys, it may contain a concatenated value; both may include partial and generic keys, etc.

If qualifying entries using a super-grouped SI-path, DBFIND must be called against the master set, although entries in all datasets in the super-group will be used to qualify the master entries.

SUPERDEX allows multiple DBFIND calls in succession to qualify entries across multiple SI-paths, datasets, and even multiple databases. A similar technique may be used to refine a selection, whereby further qualification may be performed against entries already found.

Refer to the *Qualifying entries with DBFIND* chapter of the *Programming* section for further discussion and examples.

Parameters

Base   Same as IMAGE

Dset   In addition to IMAGE, master sets as well as detail sets may be specified by name or number. If the SI-path is super-grouped, must be a master set.

                  If accessing an independent SI-path, this parameter should be left blank or set to 200.

Mode   While IMAGE allows only a single *mode* (*mode* 1) for DBFIND, SUPERDEX extends the capabilities of *mode* 1 while maintaining compatibility, and provides several additional *modes* for use on SI-paths.

Mode 1   Fully compatible with IMAGE *mode* 1. Also allows multiple values and various operators to be included in the *argument* (explained later). If no entry is found that matches the *argument*, condition word 17 ("NO ENTRY FOUND") is returned.

                  ☞ **Only DBFIND mode 1 returns the number of qualifying entries in words 5-6 of the status array.**

Mode 10   Same as *mode* 1, but does not return the number of qualifying entries in the *status* array and is therefore more efficient.

In *mode* 1, not only is the SI-pointer set to the first qualifying SI-index entry, but the B-tree is traversed to locate all qualifying SI-indices. It is more efficient to only set the SI-pointer, so *mode* 10 should be used in place of *mode* 1 whenever the number of qualifying entries is not required.

In *mode* 10, the chain entry count in the *status* array is always set to 1, for compatibility with programs that call DBFIND and test to make sure that the chain count is not zero.

Mode 100   Positions the SI-pointer <u>before</u> the first SI-index in the B-tree (i.e. before the lowest alphabetical entry in the set in ascending order). In this *mode*, the *argument* is ignored and may be left blank.

Mode 1*nn*/-1*nn*   Reads *nn* words of the *argument* and sets the SI-pointer before the first qualifying SI-index entry. If no matching entry exists, condition word 17 ("**NO ENTRY FOUND**") is returned but the SI-pointer is set immediately before the nearest-matching SI-index. If prefixed with a minus sign (−), reads *nn* bytes instead of words.

Mode 200   Positions the SI-pointer <u>after</u> the last entry, in ascending order (i.e. after the highest alphabetical entry in the set). In this *mode*, the *argument* is ignored and may be left blank.

Mode 2*nn*/-2*nn*   Reads *nn* words of the *argument* and sets the SI-pointer after the last qualifying SI-index entry. If no matching entry exists, condition word 17 ("**NO ENTRY FOUND**") is returned but the SI-pointer is set immediately after the nearest-matching SI-index. If prefixed with a minus sign (−), reads *nn* bytes instead of words.

Status   Same as IMAGE, although <u>the chain count (words 5-6) is set by *mode* 1 only</u>; in *mode* 10, the chain count is set to a constant value of 1. In relational access mode, the chain count reflects the total number of SI-indices in the active SI-subset.

The first-on-chain and last-on-chain pointers (words 7-10) are set only for IMAGE paths and not SI-paths, unless the SISETLINK JCW has been set to 1. If the SISETLINK JCW has been set, the first-on-chain and last-on-chain values will contain the correct values for the given SI-path.

Additionally, a unique internal number utilized in performing logically-related DBFINDs against multiple databases is either returned or specified in word 2, which is unused by IMAGE.

SUPERDEX returns a condition word of 0 in the first word on a successful call and a non-zero condition word on an unsuccessful call, like IMAGE; however, a SUPERDEX DBFIND with a *mode* of 1*nn* or 2*nn* may return a condition word 17 error ("**NO ENTRY FOUND**") while still setting the SI-pointer before or after the nearest qualifying entry, respectively.

Item   Specifies either an IMAGE path or an SI-path. If an IMAGE path, the name or number of the IMAGE search field is specified as usual. If an SI-path, the name or number of the SI-path or the item number of the first SI-subkey in the SI-key is specified.

☞ **If there is more than one SI-path that starts with the same item number, or if the SI-path is an independent SI-path, the SI-path name or number must be specified.**

If the SI-path number is to be used, it is recommended that the numbers **NOT** be hard-coded in the program. A DBINFO mode 312 should be called against the path to retrieve the SI-path number. SI-path numbers are assigned dynamically, based on the configuration of the database. Therefore, if SI-paths are added or deleted the SI-path number for any SI-path may change.

If both an IMAGE path and an SI-path with the same name exist in the same dataset and DBFIND is called in *mode* 1, SUPERDEX will use the SI-path instead of the IMAGE path. If the search of the SI-path is unsuccessful, the IMAGE path is used. If the IMAGE-path search is unsuccessful, the condition word is set to 17.

In performing successive DBFINDs against multiple datasets, a common item used to logically link the datasets together (called the *SI-link*) may additionally be specified. It is required that the item assigned as the SI-link be configured as an SI-subkey; alternately, for SI-paths against a master dataset, it may be the IMAGE search field.

The SI-link is separated from the SI-path name by a comma, with the combined value terminated by a **SPACE** or **;** as shown:

   SI-path, SI-link;

If the SI-path is passed as an item number rather than a name, the item number of the SI-link should be specified in the second word of the *item* array.

If performing a projection, which is used to logically link two datasets that do not contain a common item by reassigning the SI-link, the *item* parameter takes the form:

   SI-path, new SI-link;

To locate entries that have been found by previous DBFINDs in the active SI-subset rather than in the dataset, specify a null *item* of **0** or **;**.

Argument

In IMAGE, the DBFIND *argument* must specify an exact search field value. In SUPERDEX, the *argument* for DBFIND *modes 1 and 10* may contain:

- an exact SI-key value (or concatenated value)
- a single @
- a partial SI-key value with one or two @ surrounded with << >>
- a generic SI-key value containing one or more embedded ?s (the alphanumeric matchcode) or #s (the numeric matchcode)
- a partial SI-key value preceded by either the >=, <=, or <> relational operators
- a range of two or more values with embedded relational operators (e.g. >=A@<=B@)
- an ASCII value prefixed by ==, which causes the value to be converted to binary for comparison
- multiple SI-key values, with the argument beginning with the tilde (¯) and ending with the semi-colon (;), and including one or more of the boolean operators **AND, OR,** or **NOT** (specified in SQL Notation)
- multiple SI-key values, with the argument beginning with the tilde (¯) and ending with the semi-colon (;), and including one or more of the boolean operators +, ,, or ‑ (specified in Infix Notation)
- multiple SI-key values, delimited with square brackets ([ ]), and including one or more of the boolean operators &, |, or ! & (specified in Reverse Polish Notation)
- the special operators, such as /, \, and $, used for manipulating the active SI-subset and backup SI-subset
- the special operator ^, used for swapping the active and backup SI-subsets
- the special operator *, used for projection
- the special operator @@, which rewinds the virtual SI-chain, and in *mode 1* returns its entry count
- many combinations of these constructs

The following operators may be embedded in the *argument* for DBFIND *modes 1 and 10*:

Conditional operators	
@	any variable number of alphanumeric characters
?	any single alphanumeric character
#	any single numeric character

Relational operators	
>=	greater than or equal to
<=	less than or equal to
<>	not equal to

Boolean operators	
SQL Notation	
AND	and
OR	or
NOT	and not
Infix Notation	
+	and
/	or
−	and not
Reverse Polish Notation	
&	and
|	or
! &	and not

Special Operators (for successive DBFIND calls)	
&	ANDs backup SI-subset with active SI-subset, replaces active with result, deletes backup
|	ORs backup with active, moves result into active, deletes backup
! &	AND NOTs backup with active, moves result into active, does not change backup
/	moves copy of active into backup
****	moves backup into active, deletes backup
¯	swaps the active and backup
[ ]	replaces backup with active, erase active
[*]	performs a projection
@@	rewinds virtual SI-chain; returns entry count in *mode* 1

☞ In *modes* 100 and 200, the *argument* value is ignored.

Keywords If DBFIND is called against a keyworded SI-path and the length of the specified keyword exceeds the *keyword length* configured for the SI-path, the specified keyword is truncated to the keyword length and matching is done on the truncated value.

Terminator If DBFIND is called in *mode* 1 or 10 and the specified *argument* is not the full SI-key value, either the *buffer* must be padded with spaces or the *argument* value must be terminated by:

■ for alphanumeric fields with *arguments* not surrounded with << >>, a single SPACE followed by a single @

■ for alphanumeric fields with *arguments* not surrounded with << >>, a single @ (used for partial-key retrieval)

■ for alphanumeric fields with the *argument* surrounded with << >>, up to two @s.

■ for numeric fields, a single SPACE

■ for *arguments* beginning with the tilde (~), the semi-colon (;)

■ for *arguments* ending with a] or boolean operator, a single SPACE or an @

Delimiters

If an *argument* begins with a tilde (~) and ends with a semi-colon (;), or an *argument* value is enclosed in square brackets, this specifies that SUPERDEX should perform a relational retrieval:

```
->=C@;
[>=C@]
```

The tilde and semi-colon are required when using the SQL Notation or Infix Notation operators. The square brackets are required when using the Reverse Polish Notation operators. Using these relational accesses prevents entries from being returned more than once when the search criteria would qualify them multiple times.

Booleans

Boolean operations between multiple values may be specified in a DBFIND *argument* in SQL Notation, Infix Notation, or Reverse Polish Notation, as used in HP calculators. In RPN, the operator always follows the values being compared, while the other two require the operator to precede the value(s). For example:

```
~A@ or B@ or >=P@;        SQL Notation
~A@,B@,>=P@;              Infix Notation
[A@][B@] | [>=P@] |       Reverse Polish Notation
```

These locate all entries that begin with "A" or "B", or that begin with "P" or an alphabetically higher letter.

Refinement

The special operators for managing the active and backup SI-subsets may be used alone or in combination in the *argument*; if used alone, DBFIND will manage the SI-subsets but not select any new entries. Alternately, the special operator(s) may be used to prefix any *argument* value and both the selection and SI-subset management operations will be performed in the same intrinsic call.

Data types

For DBFIND *modes* 1 and 10 in indexed access, ASCII numbers may be specified for most numeric items (data types I, J, P, R, and Z) if prefixed with ==, >=, <=, or <> or, if appropriate, a - (negative sign)--the == operator simply converts an ASCII value specified to binary format for comparison.

☞ **The exception is data type K, for which values must be specified in binary.**

For concatenated SI-keys that contain SI-subkeys of mixed data types (alphanumeric and numeric) there are many different ways to specify data. Refer to the **Finding entries in a concatenated SI-key** paragraph in *Section 4*.

When performing relational access, the *argument* must be specified in ASCII. Values are automatically converted to binary for comparison with binary data values (do not prefix the argument with the == conversion operator).

For data types P and Z, SUPERDEX's DBFIND treats unsigned and positive values equivalently. For data type P, the sign is held in the last nibble (4 bits); for type Z, the sign is over-punched in the last byte.

Real numbers (items of data type R) may include embedded decimal points (.), exponential signs (E), and positive (+) and negative (-) signs.

Restrictions   A few restrictions exist in *arguments* that may be specified:

- for range searches, the first value specified (start point) must be less than the second value (end point)
- for range searches (as well as searches that use values that start with the `<=`, `>=`, or `<>` relational operators), a **?** embedded in a value is not recognized as an operator but as a regular character
- the **@** wildcard may only be specified as the last character in a value, unless the *argument* value is surrounded with the `<<` `>>` operators. Any characters that follow an **@** are ignored except when performing a range or not-equal-to retrieval
- exclusions (values preceded by `<>`) must be the last value in the *argument*

Data type Z   Additional restrictions exist in *arguments* that may be specified for SI-keys of data type Z:

- a **?** or **@** embedded in a value is not recognized as an operator but as a literal character. As the SI-key is numeric, this could result in "**ILLEGAL ASCII DIGITS**" in COBOL or as condition word 17.
- leading zeroes must be specified unless the *argument* is prefaced with the operator `==`, `<=`, `>=`, `<>` or relational access is being used.
- *mode* 1*nn* or 2*nn* must be used to qualify entries when using a partial key value for both simple and concatenated SI-keys
- for concatenated SI-keys, *mode* 1*nn* or 2*nn* must be used to qualify entries if the full SI-key value is specified
- for concatenated SI-keys that contain SI-subkeys of mixed data types and for which the first SI-subkey is numeric, a numeric value may be specified in *mode* 1 or 10 for the first SI-subkey only. Characters specified after the first SI-subkey will cause unpredictable results. Use *mode* 1*nn* or 2*nn* to qualify partial or full concatenated SI-keys

Effect of DBFIND on the SI-pointer and current path

The following table summarizes the effects of DBFIND on the SI-pointer and current SI-path:

mode	condition word = 0	condition word <> 0
1	before entry	current path not set**
10	before entry	current path not set**
100*	before first entry	current path not set
1nn	before entry	if cw = 17, before next entry
200	after last entry	current path not set
2nn	after entry	if cw = 17, after previous entry

*   may be followed by either DBGET *mode* 5 or 15 or *mode* 6 or 16 and will start at either the beginning or end of the dataset, respectively, as compatible with IMAGE

**   if using relational access, the current path is retained

DBGET intrinsic

Modes 5 and 6 return entries on a logical SI-chain rather than a physical IMAGE chain. Also, DBGET *mode* 5 and 6 work the same in both master and detail sets (in IMAGE, *mode* 5 and 6 DBGETs against a master set traverse the synonym chain, and are rarely used).

New *modes* 15 and 16 are available. These *modes* operate the same as *modes* 5 and 6, except they continue to retrieve entries even after the SI-keys no longer match the *argument*, all the way to the end or beginning of the set. Effectively, they perform greater-than-or-equal to and less-than-or-equal to retrievals, respectively.

Also, the new ı *list* construct is available for reading SI-indices only, for greater efficiency.

If the next- and previous-record numbers in the chain are required, set the JCW **SISETLINK** to 1. SUPERDEX will place the next- and previous-record numbers into the status array. This includes any DBGET *mode*, or if relational access is being executed.

Refer to the *Retrieving entries with DBGET* chapter of the *Programming* section for further discussion.

Parameters

Base	Same as IMAGE
Dset	Same as IMAGE--sets may be specified by name or number.
	If accessing an independent SI-path, this parameter should be left blank or set to **200**.
Mode	*Modes* 5 and 6 are enhanced to work on SI-paths, and new *modes* 15 and 16 permit retrieval of entries in addition to those qualified by the previous DBFIND.
Mode 4	*Mode* 4 continues to function as in IMAGE.
	If the SI-Path in the DBFIND is a one-to-one relationship index (simple index SI-path or concatenated SI-path) and no relational accessing was executed, this mode, in conjunction with modes 5 and 6, functions the same as IMAGE.
	If relational access was executed, or the SI-Path is not a one-to-one index, set the JCW **SISETLINK** to 1 and SUPERDEX will function the same as IMAGE.
Mode 5	*Mode* 5 continues to function as in IMAGE if the DBGET is performed on an IMAGE path.

Mode 6

Mode 5 against an SI-path returns all entries qualified by the previous DBFIND in ascending sorted sequential order if index access was used. Once all qualifying entries have been returned, the condition word is set to 15 ("END OF CHAIN").

Mode 6 continues to function as in IMAGE if the DBGET is performed on an IMAGE path.

Mode 6 against an SI-path returns all entries qualified by the previous DBFIND in descending sorted sequential order if indexed access is used. Once all qualifying entries have been returned, the condition word is set to 14 ("BEGINNING OF CHAIN").

Mode 15

Same as *mode* 5, but continues to retrieve entries in ascending sorted sequential order even after the SI-keys no longer match the *argument* (greater-than-or-equal-to retrieval).

Mode 16

Same as *mode* 6, but continues to retrieve entries in descending sorted sequential order even after the SI-keys no longer match the *argument* (less-than-or-equal-to retrieval).

Status

When using *modes* 5, 6, 15, and 16 against an SI-path, only the first four words of the *status* array contain accurate information, unless the JCW **SISETLINK** has been set to 1.

If the JCW **SISETLINK** has been set to 1, words 7-8 and 9-10 will be sent to the previous- and next-record numbers on the chain.

List

Same as IMAGE, but additional considerations for subsequent DBUPDATE and DBDELETE exist, and a new *list* is available.

If the **SIEXTLEN** JCW was used to configure a concatenated SI-key with more than four non-contiguous SI-subkeys, it is required that all items that were not explicitly referenced when defining the SI-key be included in the *list* in the order in which they appear in the dataset before calling DBUPDATE or DBDELETE.

! *list*

The **!** *list* returns the SI-index rather than the data entry. This is much faster than any other *list* construct because SUPERDEX needs to access only an SI-dataset.

Several functions can be accomplished with the **!** *list* by reading only the SI-index rather than the entire data entry, for example:

- pattern matching and character validation
- intermediate storage of an SI-index to reset the SI-pointer after changing to another SI-path
- checking for the existence of common SI-indices in two datasets, by alternately reading SI-paths from each set

If the corresponding data entry is needed for any SI-index (to DBUPDATE or DBDELETE the data entry, for example) use DBGET *mode* 1 (reread current entry) with a *list* other than **!** to read it.

The **!** *list* is never seen by IMAGE, so the current *list* before the **!** *list* was specified will still be active (i.e. the **✱** *list* may still be used as usual).

In reading an independent SI-path, only the ! *list* is allowed.

SI-counter

If using the **!** list, an *SI-counter* may optionally be appended as numeric literal to specify the number of SI-indices that should be returned by a single DBGET call, in the format:

! *SI-counter*

The **SI-counter** is terminated by **;** or blank

DBUPDATE

If the DBGET used to locate an entry for updating specifies neither the **@** *list* nor a *list* that includes all the SI-subkeys whose values are being changed, SUPERDEX will automatically perform one or more rereads (DBGET *mode* 1) with various *lists* when performing the DBUPDATE.

This is both inefficient and causes the current *list* to change without the knowledge of the program, and therefore any subsequent calls done without re-initializing the *list* may be faulty. It is therefore recommended that the **@** *list* or a *list* that contains all SI-subkeys be used when calling DBGET before DBUPDATE or when calling DBUPDATE.

☞ Using the **/2** database option in SIMAINT to allow two DBOPENs to be performed eliminates this *list* concern.

DBDELETE

Similarly, if all SI-keys are not included in the *list* of the DBGET used to locate an entry for deletion, SUPERDEX will automatically change the *list* to **@** (unless it is already **@** or **;**).

Because the current *list* may change and therefore any subsequent calls done without re-initializing the *list* may be faulty, it is recommended that the **@** *list* or a *list* containing all SI-keys be used when calling DBGET before DBDELETE (the **;** *list* [null list] may also be used, but is inefficient).

☞ Using the **/2** database option in SIMAINT to allow two DBOPENs to be performed eliminates this *list* concern.

Buffer   Same as IMAGE.

Argument   Ignored for *modes* 5, 6, 15, and 16.

Effect of DBGET on the SI-pointer and current path

The following table summarizes the effects of DBGET on the SI-pointer:

mode/relationship	condition word = 0	condition word <> 0
5 or 15*	before next entry	if *cw* = 15, no change
6 or 16*	after previous entry	if *cw* = 14, no change
other unique	on entry	no change
other non-unique	no change	no change

*   if switching from *mode* 5 or 15 to *mode* 6 or 16, the same entry is not returned twice

DBINFO intrinsic

Same as IMAGE with additional *modes* **311** and **312** implemented to return information about the SUPERDEX SI-path configuration for a database.

Parameters

Base   *Base-id* (same as IMAGE).

Qualifier   For *mode* **311**, name or number of the dataset for which to return SI-path information.

For *mode* **312**, name of the dataset to which the SI-path is related followed by the name of the SI-path for which to return information, delimited by *;*, as shown:

dataset name; SI-path name

Alternately, the dataset number may be specified as an integer followed by the SI-path name of up to 16 characters (with no delimiter):

dataset number SI-path name

Or, the dataset number may be specified as an integer followed by the item number of the first SI-subkey as an integer (with no delimiter):

dataset number item number

☞ **If there are multiple SI-paths related to a dataset that contain the same item as the first SI-subkey, the SI-path that appears first in the SI-definitions is returned.**

In addition, the *path number* can be used in the same format as *item number* above.

Mode   All IMAGE *modes* are acceptable.

Mode 311   *Mode* **311** is used to check for presence of SUPERDEX and to retrieve information about the SI-paths related to a specified dataset.

Mode 312   *Mode* 312 is used to determine the characteristics of a specific SI-path related to a dataset.

This *mode* is intended for use in determining whether an SI-path is keyworded or grouped and, for grouped paths, the items that are included in the group.

Status   Only the condition word is set.

Buffer   For *mode* 311, information about the related SI-paths is returned as the number of SI-paths followed by 16 words of information for each SI-path, in the following format:

word	description
1	number (count) of SI-paths
2-9	SI-path name of SI-path #1
10	item number of first SI-subkey of SI-path #1; 0 for custom index
11	length in words of first SI-subkey of SI-path #1
12	item number of second SI-subkey of SI-path #1 (0 if not configured)
13	length in words of second SI-subkey (0 if not configured)
14	item number of third SI-subkey of SI-path #1 (0 if not configured)
15	length in words of third SI-subkey (0 if not configured)
16	item number of fourth SI-subkey of SI-path #1 (0 if not configured)
17	length in words of fourth SI-subkey (0 if not configured)
18-33	same information for SI-path #2
34-49	same information for SI-path #3
...	same information for additional paths

☞ The space required for **BUFFER** is not larger than the length of the SI-item. For grouped SI-Paths the information is returned only once. Use Mode 312 to find all members of the group. For Super-grouped SI-Paths use mode 312 for all related details.

For *mode* 312, information about the SI-path is returned in the following format:

word	description
1	compound item flag (1 if IMAGE compound item, otherwise 0)
2	grouped SI-path flag (1 if grouped SI-path, 2 if super-grouped, otherwise 0)
3	keyworded SI-path flag (1 if keyworded SI-path, otherwise 0)
4	index blanks flag (1 if blank values are indexed, otherwise 0)
5	SI-path number
6	offset for first SI-subkey
7	offset for second SI-subkey (0 if not configured)
8	offset for third SI-subkey (0 if not configured)
9	length in words of SI-index
10	length in words of SI-key
11	number of SI-subkeys (0 for non-concatenated SI-keys)
12	number of SI-keys in group (0 for non-grouped SI-paths)
13	minimum keyword length (0 for non-keyworded SI-paths)
14	average number of SI-indices (0 for non-keyworded and non-custom SI-paths)
15	item number of first SI-subkey of first SI-key in group (0 for non-grouped SI-paths)
16	item number of first SI-subkey of second SI-key in group (undefined if non-grouped SI-paths)
...	for additional SI-keys in group

DBLOCK intrinsic

There are no syntax changes to the DBLOCK intrinsic.

Based on its configuration, SUPERDEX may automatically apply a dataset-level DBLOCK against one or more datasets, including the appropriate SI-dataset, when a program calls any of the following intrinsics:

- DBPUT
- DBPUTIX
- DBDELETE
- DBDELIX
- DBUPDATE

If the database is configured for a separate DBOPEN for locking, SUPERDEX calls both DBLOCK and DBUNLOCK against the appropriate SI-dataset. Refer to the _Locking_ chapter at the beginning of the _Programming_ section for a complete discussion about SUPERDEX and locking.

DBMEMO intrinsic

DBMEMO is the same as in IMAGE, but if called against a database for which logging is not enabled, it is ignored.

DBOPEN intrinsic

The DBOPEN intrinsic is the same as in IMAGE, although it establishes the run-time SUPERDEX structures and is therefore slightly slower. Additionally, it does **NOT** set the first word of the data base variable as IMAGE does. SUPERDEX stores the pointer to the SUPERDEX User area (SDU) in the first word. This then allows SUPERDEX to retrieve the pointer for the IMAGE DBU from within the SDU, thus **ALL** calls to the data base must be executed by SUPERDEX.

☞ SUPERDEX may be configured to perform two DBOPENs against a database for greater efficiency: all access and locking of datasets containing data entries are performed via the first open, while all access and locking of the SI-datasets are performed via the second open. Refer to the *Locking* chapter of the *Programming* section for a discussion.

If the SI-indices are kept in a separate SI-index base, SUPERDEX automatically DBOPENs (and DBCLOSEs) the SI-index base.

DBPUT intrinsic

Functionally the same as in IMAGE, except entries with blank SI-keys, and concatenated SI-keys for which the first SI-subkey is blank, are not indexed whereas in IMAGE they would all be placed on a null chain. This may be overridden as a configuration option.

Appropriate SI-indices, including SI-indices for super-grouped SI-paths and custom SI-indices generated by the SIUSER procedure, are automatically added into the B-trees when DBPUT is called.

LIST parameter

The *list* parameter used when calling DBPUT must include not only the IMAGE search fields but all fields configured as SI-subkeys or SI-keys. If the *list* does not include all IMAGE search fields and SI-keys, condition word -53 ("**DBPUT IS MISSING A SEARCH OR SORT ITEM**") is returned.

Locking

A DBPUT against a SUPERDEX'ed dataset may also internally access the corresponding SI-dataset to add corresponding SI-indices. This should be taken into consideration when locking, since the appropriate SI-dataset must be locked during a DBPUT, which may be done implicitly or explicitly. Refer to the *Locking* chapter of the *Programming* section for a discussion of the various locking methods.

Logical transactions

If the base is enabled for logging, SUPERDEX automatically imposes DBBEGINs and DBENDs around each DBPUT that does not already specify them. The DBEND is imposed regardless of whether or not the DBPUT is successful.

DBPUTIX intrinsic

DBPUTIX is a new SUPERDEX intrinsic used to explicitly add SI-indices into B-trees (its counterpart is the new DBDELIX intrinsic, which explicitly deletes SI-indices).

The DBPUTIX intrinsic accesses only the appropriate SI-dataset and is used to maintain independent indices. It also provides a method for adding custom SI-indices in addition to those maintained automatically by DBPUT.

Refer to the _Adding, updating, and deleting entries_ chapter of the _Programming_ section for further discussion.

Syntax   DBPUTIX _(base,dset,mode,status,item,buffer)_

The DBPUTIX intrinsic is syntactically similar to DBPUT except that the _list_ parameter is replaced by an _item_ parameter and the _buffer_ parameter contains the full SI-index, including the SI-extension.

Parameters

Base   The _base-ID_ (same as DBPUT).

Dset   Name or number of the dataset in which the corresponding data entry exists.

                  If accessing an independent SI-path, this parameter should be left blank or set to 200.

Mode   An integer with the value 1.

Status   Only the condition word is set.

Item   The name of the SI-path in whose B-tree to add the specified SI-index.

Buffer   The full SI-index value including the SI-extension, which is:

                  ■ the search field value for a master dataset
                  ■ the relative record number for a detail dataset
                  ■ a suitable user-defined value for independent SI-paths

Locking

A DBPUTIX against an SI-path internally accesses the corresponding SI-dataset to add one or more SI-indices. This should be taken into consideration when locking, since the appropriate SI-dataset must be locked during a DBPUTIX, which may be done implicitly or explicitly. Refer to the *Locking* chapter of the *Programming* section for a discussion of the various locking methods.

Logical transactions

If the base is enabled for logging, SUPERDEX automatically imposes DBBEGINs and DBENDs around each DBPUTIX that does not already specify them. The DBEND is imposed regardless of whether or not the DBPUTIX is successful. If DBPUTIX is used in combination with DBPUT, the program should include a DBBEGIN before the DBPUT and the DBEND after the DBPUTIX.

Error handling

Since SUPERDEX uses standard IMAGE messages to report all errors and exceptional conditions, the same messages that are used by DBPUT are displayed for DBPUTIX.

DBUNLOCK intrinsic

There are no syntax changes to the DBUNLOCK intrinsic.

If the database is configured for a separate DBOPEN for locking, SUPERDEX calls DBUNLOCK (after DBLOCK) against the appropriate SI-dataset. Refer to the *Locking* chapter at the beginning of the *Programming* section for a complete discussion about SUPERDEX and locking.

DBUPDATE intrinsic

Functionally the same as IMAGE, except it may also be used to change the value of an SI-key. IMAGE does not permit critical fields (search and sort fields) to be updated by DBUPDATE.

SI-indices for super-grouped SI-paths and custom SI-indices generated by the SIUSER procedure are automatically maintained by SUPERDEX when DBUPDATE is called. It is required that the full (@) *list* be used in this case.

DBOPEN mode 2

If the database is opened in *mode* 2, DBUPDATE cannot be used to update fields used in SI-keys, since this may require that an internal DBPUT and DBDELETE be performed against the corresponding SI-dataset, which cannot be done in DBOPEN *mode* 2. If this is attempted, condition word -41 (**"DBUPDATE WILL NOT ALTER A SEARCH OR SORT ITEM"**) is returned.

Locking

A DBUPDATE against a SUPERDEX'ed set may also internally access the corresponding SI-dataset to add and delete corresponding SI-indices. This should be taken into consideration when locking, since the appropriate SI-dataset must be locked during a DBUPDATE against an SI-key, which may be done implicitly or explicitly. Refer to the *Locking* chapter of the *Programming* section for a discussion of the various locking methods.

Logical transactions

If the base is enabled for logging, SUPERDEX automatically imposes DBBEGINs and DBENDs around each DBUPDATE that does not already specify them. The DBEND is imposed regardless of whether or not the DBUPDATE is successful.

DBGET used to locate entry for DBUPDATE

If the DBGET used to locate an entry for updating specified a *list* that does not include all the SI-keys in the dataset that are being changed, SUPERDEX will automatically perform one or more rereads (DBGET *mode* 1) with various *lists*. This is both inefficient and causes the current *list* to change without the knowledge of the program, and therefore any subsequent calls done without re-initializing the *list* may be faulty.

It is therefore recommended that the @ *list* or a *list* that contains all SI-subkeys be used when calling DBGET before DBUPDATE or when calling DBUPDATE; otherwise, DBUPDATE may internally change the *list*.

☞ Using the /2 database option in SIMAINT to allow two DBOPENs to be performed eliminates this *list* concern.

DBXBEGIN intrinsic

☞ Dynamic transaction intrinsics (DBXBEGIN, DBXEND, DBXUNDO) are not supported at this time.

DBXEND intrinsic

☞ Dynamic transaction intrinsics (DBXBEGIN, DBXEND, DBXUNDO) are not supported at this time.

DBXUNDO intrinsic

☞ Dynamic transaction intrinsics (DBXBEGIN, DBXEND, DBXUNDO) are not supported at this time.

SITRANSLATE intrinsic

SITRANSLATE is an procedure to translate input from Infix notation to the Reverse Polish Notation.

Infix operators are **+** (AND), **-** (AND NOT), and **,** *(comma)* (OR). The input string is evaluated from left to right, with no precedence between operators. Parentheses can be used to change the order of evaluation.

Operands containing imbedded blanks must be delimited by double quotes ("). The input string must be terminated by a blank.

The **OPERATOR** parameter allows one infix operator to be supplied programmatically. This is useful in applications prompting the user only for operands.

Parameters

Operator	*(byte)*	Values allowed are "**+**", "**-**", "**,**" and blank (no operator supplied)
Input	*(byte array)*	Input string to be translated, terminated by blank, in infix notation.
Output	*(byte array)*	Output string of translation, terminated by blank, in Reverse Polish Notation.
Error	*(integer)*	Zero (0) upon successful operation.

Examples

```
1)   OPERATOR=  blank
     INPUT    =  NEW-"NEW YORK"
     OUTPUT   =  [NEW][NEW YORK]!&

2)   OPERATOR=  blank
     INPUT    =  (COMB+BLIND),(HOLD+DRILL)-FASTEN@
     OUTPUT   =  [COMB][BLIND]&[HOLD][DRILL]&|[FASTEN@]!&

3)   OPERATOR=  +
     INPUT    =  "LOS ANGELES"
     OUTPUT   =  [LOS ANGELES]&

4)   OPERATOR=  blank
     INPUT    =  +"LOS ANGELES"
     OUTPUT   =  [LOS ANGELES]&
```

SIUSER procedure

SIUSER is an optional user-written procedure that is invoked by SUPERDEX to compute one or more custom SI-indices for entries whenever DBPUT, DBUPDATE, or DBDELETE is called and from the SIMAINT utility. It is useful for establishing SI-indices that cannot be composed of dataset fields as they are represented but which can be calculated using values in the data entry, as well as for SI-indices that require:

- more than four non-contiguous SI-subkeys
- date conversion
- reordering
- upshifting
- stripping
- other parsing

☞ ASK customers should contact Bradmark's Technical Support office for additional information.

The SIUSER procedure is written by the user and installed in an SL or XL that contains the SUPERDEX segments. Each SL or XL may contain a different SIUSER procedure, or the same SIUSER procedure may be placed in various SLs or XLs. The SIUSER procedure is invoked once for every DBPUT and DBDELETE and twice for each DBUPDATE (internally, the SI-index is deleted and re-added).

☞ The SIUSER procedure must be in the appropriate object code format for both the SUPERDEX environment and the application(s) that will be updating the custom index. This means if the application is a native-mode application, then SIUSER must be included in the SUPERDEX XL. If the application is a compatibility-mode application, the procedure must be included in the SUPERDEX SL. **The native-mode version of SIMAINT will not access the compatibility-mode version of SIUSER, and the compatibility-mode version of SIMAINT will not access the native-mode version of SIUSER.**

If multiple custom SI-paths have been configured, it is necessary to add conditional statements into the SIUSER procedure to specify which statements are executed for which SI-path.

Syntax   SIUSER *(base,dset,item,buffer,index)*

Parameters

The first four parameters are supplied by the program; the *index* value is returned by SIUSER.

Base   The *Base-id* (returned by DBOPEN).

Dset   The name or number of the dataset in which the corresponding data entry exists. When called from DBPUT, DBUPDATE, or DBDELETE, its format is the same as specified in those intrinsics. If called from the SIMAINT program, the dataset name is used.

Item   The name of the SI-path in which to add the specified SI-index.

Buffer   The full data entry (@ *list*) used in the DBPUT or DBDELETE.

Index   This is an output parameter only--its value is returned by SIUSER.

The first word contains a count of the number of SI-indices to be created, followed by their values in the length defined for the SI-path (use SIMAINT,LIST to look up the length). Up to 16 indices may be returned.

To cause SUPERDEX not to generate any indices for an entry, specify o in the first word of this parameter.

SUPERDEX will automatically add the appropriate SI-extension to form the SI-index by appending either the entry's IMAGE search field value (if a master dataset) or its relative record number (if a detail).

DBPUT, DBUPDATE, and DBDELETE

SI-indices generated by the SIUSER procedure are automatically maintained by DBPUT, DBUPDATE, and DBDELETE.

If DBPUT or DBDELETE is called with a *list* other than @, SUPERDEX will perform a reread using the @ *list*. If DBUPDATE is called with a list other than @, no update of the SI-indices will be done.

☞ Using the /2 database option in SIMAINT to allow two DBOPENs to be performed eliminates this *list* concern.

SIMAINT utility

The SIMAINT utility, when generating or reorganizing SI-indices, calls SIUSER from the group or account SL or XL in the same group in which SIMAINT is located and automatically generates the corresponding indices. For this reason, if custom SI-paths have been configured, either place a copy of the SL or XL that contains the SIUSER procedure into PUB.SUPERDEX and NOPRIV.SUPERDEX or copy SIMAINT into the group in which the SL or XL resides and run that copy.

If an SIUSER procedure is written to access an SI-path from within the procedure, the SI-path that is accessed from within the SIUSER procedure must have been configured in a previous run of SIMAINT and may not be changed in the current run. In this case, the SIMAINT must be run with ;LIB=G or ;LIB=P.

Section 6   Maintenance and utilities

This section discusses the various maintenance considerations for databases that contain SI-paths. It also reviews the various utility programs that may be used with SUPERDEX'ed databases.

Chapter 1   Database maintenance considerations
Function   Includes tables listing the various operations that require maintenance of SI-paths and the type of maintenance required.

Chapter 2   SIMAINT utility
Function   Used to reorganize and delete SI-paths. It also describes the DBLOAD, LIST, SCHEMA, and STRUCT options, and running SIMAINT in batch.

Chapter 3   SUPERDEX utility
Function   This program is used to maintain and reorganize SI-Paths in an on-line full screen environment.

Chapter 4   SIPATH utility
Function   Will display all IMAGE keys and SUPERDEX SI-Paths information in one concise display.

Chapter 5   SITEST and SIREPAIR utilities
Function   Used to check the integrity of B-trees and their correspondence to the data entries they represent, and to repair the B-trees.

Chapter 6   SICOUNT utility
Function   This utility will display the exact compression information on SI-Paths.

Chapter 7   SITRACE facility
Function   Available with SUPERDEX. This facility will trace all user IMAGE intrinsic calls, along with the SUPERDEX IMAGE intrinsic calls.

Chapter 8   SIDRIVER utility

Chapter 9   ALTPROG utility
Function   Adds capabilities to and increase MAXDATA of object program files, as required for SUPERDEX operation.

Chapter 10   SIBASE utility
Function   This utility will automatically create the separate SI-Index database.

Chapter 11   SISIZE utility
Function   This utility will modify the capacity of the SI dataset(s).

Chapter 12   QUERY/3000 utility

Database maintenance considerations

SI-indices are maintained automatically by SUPERDEX's DBPUT, DBDELETE, and DBUPDATE intrinsics. However, because SI-indices reference data entries by search field value or relative record number, certain database maintenance functions will cause the SI-indices to lose synchronization with the data entries they map and require reorganization.

Examples of this are as follows:

- error conditions, such as a system failure
- database maintenance tasks, such as reorganizing a detail dataset
- database structural modifications, such as changing the length of an item used as an SI-key
- modifications to the KWEXCLUD keyword exclusion file

Following are tables listing various database conditions that can require SI-path maintenance. |

Error and exceptional conditions that can require SI-path maintenance

Type	Description
Condition	**System failure which is not successfully recovered by ILR or DBRECOV**
Action	Run SIMAINT,DBLOAD
Condition	**Overflow of insufficient capacity in SI-dataset**
Action	Reorganize all SI-paths in the affected SI-dataset using *dataset/*R. If unsuccessful, run SIMAINT,DBLOAD
Condition	**Failed dataset erase**
Action	Erase the dataset with a utility or delete any remaining entries using QUERY.PUB.SYS, then reorganize all SI-paths related to the dataset. If unsuccessful, run SIMAINT,DBLOAD.

Database maintenance tasks that can require SI-path maintenance

Type	Description
Function	Renaming an item or dataset.
Action	Run SIMAINT (without the STRUCT entry-point) and press RETURN at the **DATASET>** prompt.
Function	Changing SI-key values in a dataset by a method other than the SUPERDEX DBPUT, DBUPDATE, or DBDELETE intrinsics.
Action	Reorganize the SI-paths in the related dataset.
Function	Any function that causes entries in a detail dataset to move to different physical locations, such as a detail dataset reorganization.
Action	Reorganize all SI-paths in the related dataset.
Function	Database DBUNLOAD/DBLOAD.
Action	Run SIMAINT,DBLOAD
Function	A database erase (such as with DBUTIL.PUB.SYS).
Action	The SUPERDEX configuration will also be erased. MUST redefine all SI-paths.
Function	Changing the length or data type of an item used in an SI-key.
Action	Run SIMAINT,STRUCT. For items which the SI-key length may not be specified (numeric), the SI-key length is readjusted automatically; otherwise, the SI-key length is unchanged unless it would exceed the item length, in which case the SI-key length is reduced.
Function	Changing the name of an item used in a SI-key.
Action	Run SIMAINT and press RETURN at the **DATASET>** prompt.
Function	Changing the name of a dataset that has at least one SI-path.
Action	Run SIMAINT and press RETURN at the **DATASET>** prompt.
Function	Deleting all SI-datasets.
Action	Delete the SI-item.

Other conditions that can require SI-path maintenance

Type	Description
Function	Modifying the keyword exclude file (KWEXCLUD).
Action	Reorganize the KWEXCLUDE SI-path and all of the keyworded SI-paths.

Redefining and reorganizing SI-paths

To redefine an SI-path or all the SI-paths related to a dataset, use the SIMAINT utility to Delete (/D) and then redefine the configuration.

To reorganize a selected SI-path or all the SI-paths related to a dataset, use the Reorganize (/R) option of the SIMAINT utility on the a specified SI-path or dataset.

To reorganize all SI-paths for a dataset or database while also regenerating the SI-definitions, run SIMAINT with the DBLOAD entry point.

DBGENERAL interface

Bradmark's general-purpose database maintenance utility, **DBGENERAL**, automatically performs the appropriate maintenance tasks against SUPERDEX structures whenever necessary (DBGENERAL version 6.0 and later).

For example, when erasing a dataset using DBGENERAL option 4.4, the corresponding SI-indices are automatically removed; when reorganizing a detail dataset using option 3.6, the corresponding SI-indices are automatically reorganized; and when renaming a dataset that has SI-paths using options 5.3 and 5.6, the internal SUPERDEX definitions are automatically updated.

Refer to the **DBGENERAL User Manual** for more information.

SIMAINT utility

The SIMAINT utility is used for both configuring new SI-paths and maintaining existing SI-paths. It also contains functions for displaying the SUPERDEX configuration for a database and regenerating the SI-definitions following structural changes to a database. These options are invoked by one of the following entry points:

- **(none)** permits SI-paths to be added, reorganized, and deleted
- **DBLOAD** reorganizes all the SI-paths for a dataset or database; regenerates the SI-definitions
- **LIST** lists all the SI-paths configured for a database
- **SCHEMA** generates a job stream to configure SI-paths based on current configuration
- **STRUCT** adjusts the SI-definitions to compensate for changes to database structure

Although new SI-paths may be defined concurrently with maintenance of existing SI-paths, only the SIMAINT functions for reorganizing and deleting SI-paths are discussed here--refer to the *Configuration/Establishing indices* section for discussions about and examples of using SIMAINT to configure new SI-paths and group existing SI-paths.

Access requirements

Before running SIMAINT, make sure:

- you have exclusive access to the database (except when using the LIST and SCHEMA options)
- you are logged on as the database creator
- you are logged into the group and account in which the database resides

It is also recommended for performance reasons that you:

- disable ILR
- disable logging
- do not run SIMAINT with `;LIB=G` or `;LIB=P`

Input rules

These rules govern SIMAINT input:

- all input may be in upper- or lower-case
- **?** displays structural help (sets and items)
- \ flushes the current response and re-prompts
- lengths are reported and specified in words, not bytes (unless otherwise specified), and it is necessary to convert for alphanumeric (data types U and X) items (e.g. X20 = 10 words).

SIEXTLEN JCW for special concatenated SI-keys

If you have configured SI-keys that contain more than four non-contiguous items by utilizing the **SIEXTLEN** JCW, it is required that this JCW be set before running SIMAINT with the STRUCT or SCHEMA entry points. To do so:

```
:SETJCW SIEXTLEN=1
```

Invoking SIMAINT

Two versions of the SIMAINT program are provided, in PUB.SUPERDEX and NOPRIV.SUPERDEX. They are identical except the **PUB** version uses techniques that require PM capability for improved speed and is therefore much faster than the NOPRIV version. The PUB version should be used to obtain the best performance.

Use the **NOPRIV** version, if you do not want to run the software with PM capability and on datasets whose block sizes are not multiples of 128 words (128, 256, 384, 512, etc.).

To invoke SIMAINT:

```
:RUN SIMAINT.PUB.SUPERDEX

SIMAINT VERSION 3.1 (06DEC91) COPYRIGHT DR. MATT / IABG (1988,1991)
```

☞ SIMAINT is run without **;LIB=G** or **;LIB=P**.

Specifying the database

Specify the name of a database, as shown:

```
DATABASE > OEDB
```

☞ The *open* characteristic with which the database was optionally defined is automatically retained and need not be re-specified. However, it may be explicitly changed from single open to separate open by appending **/1** or **/2** to the base name (but not to or from **/3**).

Specifying datasets

After specifying the database name, a list of the datasets that contain SI-paths is displayed:

```
SI-PATHS EXIST FOR THE FOLLOWING DATASETS:
PRFD
CUST/1
SHIP
ORDM
PART
ITEM
SSHIP
   - blank -
ENTER NAME OF SET TO BE MODIFIED OR NEW NAME
```

SIMAINT can be run against datasets that already have related SI-paths or those for which SI-paths have not yet been configured. If a dataset already contains SI-paths, they are displayed.

Enter the name of a manual master or detail dataset in the current database that contains SI-paths, optionally followed by one of the following suffixes:

  /D   Delete all dataset's related SI-paths
  /R   Reorganize all dataset's related SI-paths

This command instructs SIMAINT to reorganize all the SI-paths related to the CUST dataset:

```
DATASET > CUST/R
```

☞ The SI-dataset suffix (/1 through /7) with which the dataset was optionally defined is automatically retained and may not be overridden (to reassign a dataset's SI-indices to the root SI-dataset or any other SI-dataset, it is necessary to delete and redefine all related SI-paths).

Specifying SI-paths

All SI-paths related to the specified dataset and their attributes are displayed, as shown:

```
SI-PATHS EXIST FOR THE FOLLOWING DATASETS AND ITEMS:
CUNAME/K          CUNAME        L = 6
CUADD/K           CUADD1        L = 5
CUADD/K           CUADD2        L = 5
CUPHN             CUPHN1        L = 1
CUPHN             CUPHN2        L = 1
CUPHN             CUPHN3        L = 1
ENTER SI-PATH WITH OPTION /D /R /G OR NEW NAME
```

Enter the name of one of the SI-paths shown appended by one of the following suffixes:

/D   Delete specified SI-path
/G   Group specified SI-path (refer to the *Configuration/Establishing indices* section)
/R   Reorganize specified SI-path

Reorganizing SI-paths

SI-paths should be reorganized periodically to maintain optimum performance, and must be reorganized after certain database maintenance operations. SI-paths may be selected individually or by related dataset. Alternately, all the SI-paths for a dataset or database may be reorganized by running SIMAINT with the DBLOAD option, which also regenerates the SI-definitions.

Refer to the tables near the beginning of this section for complete details on what database maintenance tasks require SI-path reorganization.

For super-grouped SI-paths, the SI-indices for all SI-paths configured in the super-group are automatically reorganized whenever any SI-path in the super-group is reorganized.

This example specifies the reorganization of all the SI-paths related to the CUST dataset:

```
DATASET > CUST/R
DATASET >
```

SI-paths may alternately be specified individually; for keyworded SI-paths, the average number of indices may be changed, as shown:

```
DATASET > CUST
SI-PATH > CUNAME/R
ENTER AVERAGE NUMBER OF INDICES PER ENTRY > 4
SI-PATH >
```

In this example, the keyworded SI-path *CUNAME* is being reorganized by suffixing it with **/R**, and the number of indices per entry is being changed to 4.

Deleting SI-paths

SI-paths may be selected for deletion individually or by related dataset.

For super-grouped SI-paths, all SI-paths configured in the super-group are automatically deleted whenever any SI-path in the super-group is deleted.

In this example, all the SI-paths related to the SHIP dataset are deleted:

```
DATASET > SHIP/D
DATASET >
```

In this example, the SI-path *SHNAME* is being deleted by suffixing it with **/D**:

```
DATASET > SHIP
THE FOLLOWING SI-PATHS AND ITEMS ARE DEFINED:
SHIP            SHNAME        L = 6
ENTER SI-PATH WITH OPTION /D /R /G OR NEW NAME
SI-PATH > SHNAME/D
SI-PATH >
```

DBLOAD Entry Point

The **DBLOAD** entry point is used to reorganize all the SI-paths for a dataset or database. This entry point is recommended following an operation in which data entries are relocated in the database, such as a DBUNLOAD and DBLOAD. It is also useful for giving users the ability to safely and easily reorganize existing SI-paths--especially untrained users and those in turnkey environments--since only the base name needs to be specified.

Additionally, SIMAINT,DBLOAD regenerates the SUPERDEX configuration for a database, and is required after certain database maintenance functions. A complete list of database maintenance operations that require the use of SIMAINT,DBLOAD appears in tables near the beginning of this section.

In this example, all the SI-paths for the OEDB database are reorganized:

```
:RUN SIMAINT.PUB.SUPERDEX,DBLOAD
SIMAINT VERSION 3.1 (06DEC91) COPYRIGHT DR. MATT / IABG (1988,1991)


DATABASE > OEDB
DBLOAD:    562 B-TREE RECORDS DELETED        CPU 0:00:07.6 Elapsed 0:00:12
SI-PATHS EXIST FOR THE FOLLOWING SETS:
   - blank -
CUSTOMERS
ORDER-HEADERS
ORDER-LINES
ENTER NAME OF SET TO BE MODIFIED OR NEW NAME
DATASET > return
PROCESSING SI-PATH KWEXCLUDE        OF
PROCESSING SI-PATH CUSTOMER-NAME    OF CUSTOMERS         # OF ENT: 1003
        INPUT:     1003 RECORDS   100 %   CPU 0:00:01.2 Elapsed 0:00:02
        SORT:      1003 INDICES           CPU 0:00:00.0 Elapsed 0:00:00
        OUTPUT:    1003 INDICES   100 %   CPU 0:00:00.9 Elapsed 0:00:01
PROCESSING SI-PATH CUSTOMER-NAME-KW OF CUSTOMERS         # OF ENT: 1003
        INPUT:     1003 RECORDS   100 %   CPU 0:00:02.7 Elapsed 0:00:03
        SORT:      2803 INDICES           CPU 0:00:00.0 Elapsed 0:00:00
        OUTPUT:    2788 INDICES   100 %   CPU 0:00:01.6 Elapsed 0:00:04
PROCESSING SI-PATH ADDRESS1-CITY-KW OF CUSTOMERS         # OF ENT: 1003
        INPUT:     1003 RECORDS   100 %   CPU 0:00:04.2 Elapsed 0:00:06
        SORT:      4448 INDICES           CPU 0:00:00.0 Elapsed 0:00:00
        OUTPUT:    4417 INDICES   100 %   CPU 0:00:02.6 Elapsed 0:00:03
PROCESSING SI-PATH CUSTOMER-NUMBER  OF ORDER-HEADERS     # OF ENT: 2620
        INPUT:     2620 RECORDS   100 %   CPU 0:00:02.3 Elapsed 0:00:04
        SORT:      2620 INDICES           CPU 0:00:00.0 Elapsed 0:00:00
        OUTPUT:    2620 INDICES   100 %   CPU 0:00:01.4 Elapsed 0:00:02
PROCESSING SI-PATH ORDER-TYPE       OF ORDER-HEADERS     # OF ENT: 2620
        INPUT:     2620 RECORDS   100 %   CPU 0:00:02.2 Elapsed 0:00:03
        SORT:      2620 INDICES           CPU 0:00:00.0 Elapsed 0:00:00
        OUTPUT:    2620 INDICES   100 %   CPU 0:00:01.3 Elapsed 0:00:02
PROCESSING SI-PATH ORDER-PART       OF ORDER-LINES       # OF ENT: 9272
        INPUT:     9272 RECORDS   100 %   CPU 0:00:08.7 Elapsed 0:00:17
        SORT:      9272 INDICES           CPU 0:00:00.1 Elapsed 0:00:00
        OUTPUT:    9272 INDICES   100 %   CPU 0:00:06.7 Elapsed 0:00:08
PROCESSING SI-PATH PART-ORDER       OF ORDER-LINES       # OF ENT: 9272
        INPUT:     9272 RECORDS   100 %   CPU 0:00:08.8 Elapsed 0:00:10
        SORT:      9272 INDICES           CPU 0:00:00.1 Elapsed 0:00:00
        OUTPUT:    9272 INDICES   100 %   CPU 0:00:06.9 Elapsed 0:00:08
TOTAL TIME :                             CPU 0:01:03.0 Elapsed 0:01:42


END OF PROGRAM
```

There are three function keys available from this screen:

Custom Path Screen Function Keys		
Key	Label	Description
F1	HELP	Displays the Help Screen.
F2		(Not Defined)
F3	PATH SCREEN	Returns to the Path Screen, without accepting the data currently displayed on the screen.
F4		(Not Defined)
F5		(Not Defined)
F6		(Not Defined)
F7		(Not Defined)
F8	EXIT	Exits back to the Main Menu.

Once all of the data has been keyed, press the **ENTER** key. The process will then return to the Path Screen.

When running SIMAINT,SCHEMA, you are prompted for the base name and name of a file in which to write the job stream as shown:

```
:RUN SIMAINT.PUB.SUPERDEX,SCHEMA

SIMAINT VERSION 3.1 (06DEC91) COPYRIGHT DR. MATT / IABG (1988,1991)

DATABASE > OEDB
SIMAINT Job File Generation

Enter NEW filename for SIMAINT job > OEDBSI

:END OF PROGRAM
```

If the name of an existing file is specified, you are prompted for whether or not to overwrite it; otherwise, a new file is created and the job stream is written into it, as shown:

```
!JOB MGR.SUPERDEX,DEMO
!COMMENT Patch PASSWORD if necessary !
!COMMENT TimeStamp : THU, DEC 12, 1991, 10:46 AM!
RUN SIMAINT.PUB.SUPERDEX;INFO="*"
<BASE                          >OEDB
<DATASET                       >
<SI-PATH                       >KWEXCLUDE
<KW LENGTH                     >4
<SI-PATH                       >//
<DATASET                       >CUSTOMERS
<SI-PATH                       >CUSTOMER-NAME
<ITEM LEN=15 - SHORTER LEN     >-30
<OFFSET (IN BYTES)             >1
<ITEM 2                        >//
<SI-PATH                       >CUSTOMER-NAME-KW/K
<ITEM 1                        >CUSTOMER-NAME
<ITEM LEN=15 - KW LENGTH       >4
<MIN CHARS PER KW              >1
<AVERAGE KEYWORDS PER ENTRY    >6
<ITEM 2                        >//
```

Below is a description of all of the fields:

Path Display Screen Fields	
Label	**Description**
Dataset:	Current dataset being accessed
Selection Order:	Selection order of dataset
SI-Path Name:	Name of the SI-path
Path Type:	Full description of the type of path
Key Length:	Byte length of the SI-key
Number of Keys:	Total number of IMAGE items from the dataset that are included in this SI-path
Min Key Word Length:	Minimum number of characters required to create a keyworded index
Index Length:	Total length of the SI-index (includes the SI-key and SI-extension) in bytes
Avg No Key Words:	The defined average number of keywords per record
S1, S2, S3, S4	These designations are used to identify the items that are included in a concatenated SI-path. S1 is for SI-subkey-1, S2 for SI-subkey-2, etc. If the SI-path is not concatenated, none of the designations will be displayed.
Items	Names of the IMAGE items included in the SI-path
Type	The IMAGE item type (as returned by DBINFO *mode* 102)
Length	The IMAGE item length (as returned by DBINFO *mode* 102)

The function keys available from this screen are:

Path Display Screen Function Keys		
Key	**Label**	**Description**
F1	HELP	Displays the Help Screen.
F2		(Not Defined)
F3		(Not Defined)
F4		(Not Defined)
F5		(Not Defined)
F6	NEXT SCREEN	Displays the next screen of items
F7	PREV SCREEN	Displays the previous screen of items
F8		(Not Defined)

To return to the **Path Screen**, press the **ENTER** key.

STRUCT Entry Point

The **STRUCT** entry point causes SIMAINT to compare the SUPERDEX configuration with the database structure and correct any inconsistencies found. It is used after making certain structural changes to a database, but is automatically invoked by DBGENERAL. It should, however, not be run after performing a DBUNLOAD/DBLOAD--the SIMAINT program should be run with the DBLOAD option instead.

Refer to the tables near the beginning of this section for a list of maintenance functions that require the use of the STRUCT option (you will note that the STRUCT option is never used after renaming sets or items).

This example shows SIMAINT,STRUCT being used after changing the length of an item used as an SI-key:

```
:RUN SIMAINT.PUB.SUPERDEX,STRUCT

SIMAINT VERSION 3.1 (06DEC91) COPYRIGHT DR. MATT / IABG (1988, 1991)

DATABASE > OEDB
DATASET > return
```

Running SIMAINT in batch

SIMAINT can be run in batch, and uses similar dialog as the on-line. The method for creating a job stream by which to run SIMAINT in batch is to run SIMAINT with the ,STRUCT entry-point.

The discrepancies between on-line and batch use is:

- some prompts are asked in batch at all times, while on-line the prompts are asked depending on previous answers
- a line containing only a SPACE is represented in batch by a blank line
- a line containing only a RETURN (which is normally specified in a batch job as a blank line) should is represented by a line containing a double slash (//) in the first two character positions

SIMAINT will QUIT (not TERMINATE) normally upon encountering any error in batch, permitting testing of the system JCW.

For existing SI-Paths, the items already included will be displayed with an appropriate letter designation. These **CANNOT** be removed or changed. Only new items can be marked.

If creating a Supergroup SI-Path for a Detail dataset only one item can be selected (using **A**). If there are no related Master dataset(s) with the same SI-Path, the process will advance to the `Related Masters Menu`.

The function keys available from this screen are:

ITEM Screen Function Keys		
Key	Label	Description
F1	HELP	Displays the `Help Screen`.
F2	DEFINE SI-ITEMS	Advances to the `Item Definition Screen`
F3	PATH SCREEN	Returns to the `Path Screen`, without updating the item data
F4		(Not Defined)
F5		(Not Defined)
F6	NEXT SCREEN	Displays the next screen of items
F7	PREV SCREEN	Displays the previous screen of items
F8		(Not Defined)

To update the data, press the **ENTER** key. If the items are not numeric, you must also press the **F2** key to define the item specific information.

Function Key Operation

The function keys are used extensively in the SUPERDEX program to allow for easy movement through the screens. The values of each function key will change between screens, and depending on available options, on the same screen.

There are two function keys that are always the same throughout the SUPERDEX program; **F1** and **F8**. **F1** is the HELP function key and can be accessed at any time on any screen. **F8** is the EXIT function key. Pressing this key will return control back to the Main Menu screen, or if at the Main Menu screen, it will exit the SUPERDEX program.

Main Menu

```
SUPERDEX VERSION 3.1    BRADMARK TECHNOLOGIES, INC. (c)BRADMARK 1987, 1992

    1      INSTALLATION          2      PATH MAINTENANCE


    1.1    Install SUPERDEX      2.1    Maintain SI-Path(s)
    1.2    Build SI-Index Base   2.2  > Generate Path File
    1.3    Add DS/MR to Prog.    2.3  > Reorg All SI-Paths
                                 2.4  > Change SI Capacity


    3      DIAGNOSTICS           4      OTHER FEATURES


    3.1    General Path Info     4.1  > Command Driven Maint.
    3.2    Detail Path Info
    3.3    Test SI-Path(s)
    3.4  > Repair SI-Path(s)
    3.5    Compression Info
    3.6  > Structural Change



              Please enter the number
              of the desired selection: __
                                              > Exclusive Access
```

The Main Menu provides access to most of the programs in the SUPERDEX environment. Each can be accessed by entering the number associated with the particular program (**1.1** through **4.1**). Upon completion of the selected program, control will return back to this point. Refer to the other chapters in this section for more information on each of the programs.

There are only two function keys available on this screen. **F1** accesses the HELP Screen and **F8** will EXIT the SUPERDEX program.

Once the option has been keyed, press the **ENTER** key.

The `Average Number of Keywords:` is used to calculate the size of the sort file and the worst case capacity for the SI datasets. The default is set to the size of the field, divided by seven (7), always rounded up. For example, a CUSTOMER-NAME item defined in IMAGE as an X30 would have a default of 5 (30 divided by 7 equals 4.29, then rounded up to 5). A 60 character description item would be defaulted to 9 (60 divided by 7 equals 8.57, rounded to 9). This value must be 16 or less.

The function keys available from this screen are:

Item Definition Screen Function Keys		
Key	Label	Description
F1	HELP	Displays the `Help Screen.`
F2		(Not Defined)
F3	ITEM SCREEN	Returns to the `Item Screen`, without updating the item data
F4		(Not Defined)
F5		(Not Defined)
F6		(Not Defined)
F7		(Not Defined)
F8	EXIT	Exits back to the `Main Menu`

To update the data, press the **ENTER** key. Once the data has been updated, the process will return to the `Path Screen.`

Dataset Menu

```
AUG 28, 1991                SUPERDEX BASE MENU                      10:53 AM



   Please enter base name: _____  Password(Dflt=Creator): _____
   SUPERDEX Open parameter (1(Default),2,3): _


 __ _ *XXXXXXXXXXXXXXXX __ _ *XXXXXXXXXXXXXXXX __ _ *XXXXXXXXXXXXXXXX
 __ _ *XXXXXXXXXXXXXXXX __ _ *XXXXXXXXXXXXXXXX __ _ *XXXXXXXXXXXXXXXX
 __ _ *XXXXXXXXXXXXXXXX __ _ *XXXXXXXXXXXXXXXX __ _ *XXXXXXXXXXXXXXXX
 __ _ *XXXXXXXXXXXXXXXX __ _ *XXXXXXXXXXXXXXXX __ _ *XXXXXXXXXXXXXXXX
 __ _ *XXXXXXXXXXXXXXXX __ _ *XXXXXXXXXXXXXXXX __ _ *XXXXXXXXXXXXXXXX
 __ _ *XXXXXXXXXXXXXXXX __ _ *XXXXXXXXXXXXXXXX __ _ *XXXXXXXXXXXXXXXX
 __ _ *XXXXXXXXXXXXXXXX __ _ *XXXXXXXXXXXXXXXX __ _ *XXXXXXXXXXXXXXXX
 __ _ *XXXXXXXXXXXXXXXX __ _ *XXXXXXXXXXXXXXXX __ _ *XXXXXXXXXXXXXXXX
 __ _ *XXXXXXXXXXXXXXXX __ _ *XXXXXXXXXXXXXXXX __ _ *XXXXXXXXXXXXXXXX
 __ _ *XXXXXXXXXXXXXXXX __ _ *XXXXXXXXXXXXXXXX __ _ *XXXXXXXXXXXXXXXX
 __ _ *XXXXXXXXXXXXXXXX __ _ *XXXXXXXXXXXXXXXX __ _ *XXXXXXXXXXXXXXXX
 __ _ *XXXXXXXXXXXXXXXX __ _ *XXXXXXXXXXXXXXXX __ _ *XXXXXXXXXXXXXXXX
 __ _ *XXXXXXXXXXXXXXXX __ _ *XXXXXXXXXXXXXXXX __ _ *XXXXXXXXXXXXXXXX
 __ _ *XXXXXXXXXXXXXXXX __ _ *XXXXXXXXXXXXXXXX __ _ *XXXXXXXXXXXXXXXX
   "RR" - Reorganize all paths for dataset
   "DD" - Delete all paths for dataset
SUPERDEX (C) Bradmark Technologies, Inc. 1991
```

The Dataset Menu is actually a subscreen of the Base Menu. Once a database has been selected, all the datasets in the database are displayed. There are three sections of columns that include dataset names and options.

The first column is to select a dataset for maintenance during this execution. To select a dataset, enter one or two letters (A-Z) in the two character field. The datasets are processed in the alphabetical order of this two character field (i.e. A is processed before B and Z is processed before AA).

There are two reserved designations; RR and DD. The RR is used to specify that all of the SI-Paths in the dataset should be reorganized. The DD is used to specify that all of the SI-Paths in the dataset should be deleted.

The second column is used to specify which SI dataset (valid values: 1-7, or the default-blank) should be used when adding SI-Paths. If the dataset already contains at least one SI-Path, this field must be left empty.

The third column is a display-only field used to denote whether the dataset already has existing SI-Paths. If the * (asterisk) is displayed, the dataset has at least one SI-Path defined for it. If this field is blank, the dataset has no SI-Paths associated with it.

Grouped Item Offset Screen

```
Aug 28, 1991          SUPERDEX GROUPED ITEM OFFSET SCREEN          10:53 AM


    Dataset: _____  Selection Order: _
    SI-Path Name: _____
    SI-Subkey1: _____  Key Length: _____  Start:_____

STARTING              STARTING                 STARTING
POSITION    ITEM      POSITION      ITEM        POSITION     ITEM

  __   _____   __   _____   __   _____
  __   _____   __   _____   __   _____
  __   _____   __   _____   __   _____
  __   _____   __   _____   __   _____
  __   _____   __   _____   __   _____
  __   _____   __   _____   __   _____
  __   _____   __   _____   __   _____
  __   _____   __   _____   __   _____
  __   _____   __   _____   __   _____
  __   _____   __   _____   __   _____
  __   _____

SUPERDEX (C) Bradmark Technologies, Inc. 1991
```

The Grouped Item Offset Screen is used to accept the offset (starting position in characters) of each grouped item. This screen is only displayed for those grouped paths that are of an alphanumeric type.

Each grouped item has a default offset of 1, which can be either accepted or overridden. The maximum offset for any item cannot excede the length of the item itself.

For existing paths that are having additional items grouped to them, this screen will only display the new items being grouped (currently existing grouped items will not be displayed).

There are two function keys available for this screen.

Grouped Item Offset Screen Function Keys		
Key	Label	Description
F1	HELP	Displays the Help Screen.
F2		(Not Defined)
F3		(Not Defined)
F4		(Not Defined)
F5		(Not Defined)
F6		(Not Defined)
F7		(Not Defined)
F8	EXIT	Exits back to the Main Menu.

Once all of the data has been keyed, press the ENTER key . The process will then return to the PATH screen.

Path Screen

```
AUG 28, 1991                 SUPERDEX PATH SCREEN                  10:53 AM

      Dataset: _____   Selection Order: __
      SI-Path Name: _____   Action:  _ Delete _ Reorg _ Display
      Path Type: _ Group _ Supergroup _ Keyword _ Blank _ Custom
                 *(Default Path Type is Simple/Concatenated)*


Existing Paths:


  XXXXXXXXXXXXXXXX   XXXXXXXXXXXXXXXX   XXXXXXXXXXXXXXXX   XXXXXXXXXXXXXXXX
  XXXXXXXXXXXXXXXX   XXXXXXXXXXXXXXXX   XXXXXXXXXXXXXXXX   XXXXXXXXXXXXXXXX
  XXXXXXXXXXXXXXXX   XXXXXXXXXXXXXXXX   XXXXXXXXXXXXXXXX   XXXXXXXXXXXXXXXX
  XXXXXXXXXXXXXXXX   XXXXXXXXXXXXXXXX   XXXXXXXXXXXXXXXX   XXXXXXXXXXXXXXXX
  XXXXXXXXXXXXXXXX   XXXXXXXXXXXXXXXX   XXXXXXXXXXXXXXXX   XXXXXXXXXXXXXXXX
  XXXXXXXXXXXXXXXX   XXXXXXXXXXXXXXXX   XXXXXXXXXXXXXXXX   XXXXXXXXXXXXXXXX
  XXXXXXXXXXXXXXXX   XXXXXXXXXXXXXXXX   XXXXXXXXXXXXXXXX   XXXXXXXXXXXXXXXX
  XXXXXXXXXXXXXXXX   XXXXXXXXXXXXXXXX   XXXXXXXXXXXXXXXX   XXXXXXXXXXXXXXXX
  XXXXXXXXXXXXXXXX   XXXXXXXXXXXXXXXX   XXXXXXXXXXXXXXXX   XXXXXXXXXXXXXXXX
  XXXXXXXXXXXXXXXX   XXXXXXXXXXXXXXXX   XXXXXXXXXXXXXXXX   XXXXXXXXXXXXXXXX
  XXXXXXXXXXXXXXXX   XXXXXXXXXXXXXXXX   XXXXXXXXXXXXXXXX   XXXXXXXXXXXXXXXX
  XXXXXXXXXXXXXXXX   XXXXXXXXXXXXXXXX   XXXXXXXXXXXXXXXX   XXXXXXXXXXXXXXXX
  XXXXXXXXXXXXXXXX   XXXXXXXXXXXXXXXX   XXXXXXXXXXXXXXXX   XXXXXXXXXXXXXXXX


SUPERDEX (C) Bradmark Technologies, Inc. 1991
```

The **Path Screen** is the primary processing screen for the SI-Paths. It is used to execute several functions, including creating new SI-Paths, deleting SI-Paths, reorganizing SI-Paths and maintaining SI-Paths. This screen is not used to maintain the SI-Paths for the ****SPECIAL**** dataset (refer to the **Special Path Screen**).

The first two fields are display only fields. They display the current dataset being processed and the selection order value entered from the **Dataset Menu**.

The next field is the **SI-Path Name:**. This field is used to enter the name of either an existing SI-Path or a new SI-Path. If the name entered is an existing SI-Path, only the **Action:** options and the **Group** option are available. If the name entered is not an existing SI-Path, the **Action:** options is not available. If you wish to modify the **Path Type:** on an existing SI-Path (other than **Group**), it must first be deleted and then added back.

The **Action:** options are next. As stated earlier, these options are only available for existing SI-Paths. The **Delete** option will mark the SI-Path to be deleted. The **Reorg** option will mark the SI-Path to be reorganized, and the **Display** option will advance to the **Path Display Screen**. To select one of these options enter any character, such as **X**, in the field. If any of these options are selected, no other options can be selected.

The `Group Name:` is used to enter the group name of where the database is located. It is set to either the current logon GROUP or to the qualified database group name from the `Database Menu`, if it was specified. Again, the `Password:` field is used to enter the group logon password, if necessary.

The `Account Name:` is displayed, and can not be modified. If an ACCOUNT password exist, it must be entered in the `Password:` field.

☞ If this is the first SUPERDEX modification made to the database there will be questions about capacities for the SI datasets during the maintenance process. The processes will use the default values for the capacities calculated by SIMAINT. These are always the worst case calculation based on the capacity of the user dataset(s) with the SUPERDEX indices, therefore none of the options selected will fail because of a full SI dataset. Once the process has completed, an adjustment to the capacity of the SI dataset(s) may be done.

The function keys available from this screen are:

Path Display Screen Function Keys		
Key	Label	Description
F1	HELP	Displays the `Help Screen`.
F2	DATA SCREEN	Returns to the `Dataset Menu` for addition maintenance
F3		(Not Defined)
F4		(Not Defined)
F5		(Not Defined)
F6		(Not Defined)
F7		(Not Defined)
F8	EXIT	Returns back to the `Main Menu` and does not save the entered information

To update the data, press the **ENTER** key. If on-line was chosen, the process will run the maintenance program, SIMAINT, displaying the progress as it processes, and then return to the `Main Menu`. If the selection to execute the process in batch immediately was chosen, the job file will be streamed and the process will return to the `Main Menu`. If the choice was made to save the job file, the information will be saved and the process will return to the `Main Menu`.

Special Path Screen

```
AUG 28, 1991            SUPERDEX SPECIAL PATH SCREEN               10:53 AM

    Dataset: _____   Selection Order: __
    SI-Path Name: _____   Action: _ Delete
    Path Length (2/63): __        Use NLS-Sorting (Y/N): _
                                   (answer only if * is displayed)


Existing Paths:
  Len Path                 Len Path                 Len Path


 *  ## XXXXXXXXXXXXXXXX  *  ## XXXXXXXXXXXXXXXX  *  ## XXXXXXXXXXXXXXXX
 *  ## XXXXXXXXXXXXXXXX  *  ## XXXXXXXXXXXXXXXX  *  ## XXXXXXXXXXXXXXXX
 *  ## XXXXXXXXXXXXXXXX  *  ## XXXXXXXXXXXXXXXX  *  ## XXXXXXXXXXXXXXXX
 *  ## XXXXXXXXXXXXXXXX  *  ## XXXXXXXXXXXXXXXX  *  ## XXXXXXXXXXXXXXXX
 *  ## XXXXXXXXXXXXXXXX  *  ## XXXXXXXXXXXXXXXX  *  ## XXXXXXXXXXXXXXXX
 *  ## XXXXXXXXXXXXXXXX  *  ## XXXXXXXXXXXXXXXX  *  ## XXXXXXXXXXXXXXXX
 *  ## XXXXXXXXXXXXXXXX  *  ## XXXXXXXXXXXXXXXX  *  ## XXXXXXXXXXXXXXXX
 *  ## XXXXXXXXXXXXXXXX  *  ## XXXXXXXXXXXXXXXX  *  ## XXXXXXXXXXXXXXXX
 *  ## XXXXXXXXXXXXXXXX  *  ## XXXXXXXXXXXXXXXX  *  ## XXXXXXXXXXXXXXXX
 *  ## XXXXXXXXXXXXXXXX  *  ## XXXXXXXXXXXXXXXX  *  ## XXXXXXXXXXXXXXXX
 *  ## XXXXXXXXXXXXXXXX  *  ## XXXXXXXXXXXXXXXX  *  ## XXXXXXXXXXXXXXXX

SUPERDEX (C) Bradmark Technologies, Inc. 1991
```

This screen is the processing screen for ****SPECIAL**** SI-Paths. ****SPECIAL**** SI-Paths include the Independent SI-Paths and the keyword exclude SI-Path, KWEXCLUDE (see *Section 3* for more information). It is used to create or delete ****SPECIAL**** SI-Paths.

The first two fields are display only fields. They display ****SPECIAL**** for the dataset name and the selection order value entered from the Dataset Menu.

The next field is the SI-Path Name:. This field is used to enter the name of either an existing SI-Path or a new SI-Path. If the name entered is an existing SI-Path, only the Action: Delete option can be selected. If the name entered is not an existing SI-Path the Action: option is not available. If you wish to modify an existing SI-Path, it must first be deleted and then added back.

The Action: Delete option is next. As stated earlier, this option is only available for existing SI-Paths. To mark the path for deletion, key any character, such as X, in the field.

The next two fields, Path Length and NLS-sorting, are for new SI-Paths. The Path Length is used to define the entire length, in words, for the new path, and can be any numeric value between 2 and 63. The NLS-Sorting option will only accept a value if an * (asterisk) is displayed next to the field. The asterisk will only be displayed if the database has been marked for Native-Language, other than English. If the NLS collating sequence should be used to sort the indices, type Y, otherwise the default N will be used. **Only select NLS if the independent path will contain alphanumeric data.** If it will contain only binary numeric values, use the default of **N**.

SIPATH Example

```
:RUN SIPATH.PUB.SUPERDEX

SIPATH Version 3.1 (12DEC91) Copyright Bradmark Technologies, Inc.
DATABASE>ORDB.DEMO.SUPERDEX
PASSWORD>
IMAGE (Y/N)> Y


NUMBER OF DATA SETS    4

                      *** IMAGE KEYS ***

*** DATA SET ***   *** SET TYPE ***   **  KEY/PATHS **   DATA    KEY
                                                         TYPE   LNGTH


CUSTOMERS          MANUAL MASTER      CUSTOMER-NUMBER      I       2
ORDER-HEADERS      MANUAL MASTER      ORDER-NUMBER         I       2
ORDER-LINES        DETAIL DATASET     !ORDER-NUMBER        I       2
SI                 DETAIL DATASET     NO PATHS



                      *** SUPERDEX KEYS ***

*** DATA SET *** *** SI-PATH ***   TYPE  KEY    INDEX NO OF MIN KW AVE NO
                 - ITEM(S)               LENGTH LENGTH KEYS LENGTH  KWDS

CUSTOMERS
SI-PATH# = 10002 CUSTOMER-NAME      SIM    30     34     1
                 - CUSTOMER-NAME    X      30
                     (OFFSET =    1   SUBKEY LENGTH =   30)
SI-PATH# = 10003 CUSTOMER-NAME-KW   K       8     12     1      1      6
                 - CUSTOMER-NAME    X      30
                     (OFFSET =    1   SUBKEY LENGTH =    8)
SI-PATH# = 10004 ADDRESS1-CITY-KW   GK      8     12     2      1      4
                 - ADDRESS-1        X      26
                     (OFFSET =    1   SUBKEY LENGTH =    8)
                 - ------------------
                 - CITY             X      16
SI-PATH# = 10005 CUSTOMER-NUMBER    SIM     4      4     1
                 - CUSTOMER-NUMBER  I       2
                     (OFFSET =    1   SUBKEY LENGTH =    4)
```

Custom Path Screen

```
AUG 28, 1991              SUPERDEX CUSTOM PATH SCREEN              10:53 AM


     Dataset: _____     Selection Order: __
     SI-Path Name: _____
     SI-KEY Length (2/63): __
     Average number of indexes per record (1/16): __
                     Use NLS-sorting (Y/N): _
                     (answer only if *** is displayed)










SUPERDEX (C) Bradmark Technologies, Inc. 1991
```

The Custom Path Screen is used to enter the information for a custom path (set *Section 3* for more information). Since a custom path does not have any IMAGE items, this screen is used instead of the Item Screen used for regular SI-Paths.

The first three fields are display only fields. They display the current dataset being processed, the selection order value entered from the Dataset Menu, and the name of a new SI-Path from the Path Screen.

The next field, SI-KEY Length is used to define the length, in words, for the new path not including the SI-Extension (see *Appendix B* for more information), and can be any numeric value between 2 and 63, inclusive.

Next is the Average number of indexes per record field. This is used to calculate the proper capacity for the corresponding SI dataset and to calculate the proper size for sorting the indexes. The value can be any numeric value from 1 to 16.

Finally, the NLS-Sorting option will only accept a value if an * (asterisk) is displayed next to the field. The asterisk will only be displayed if the database has been marked for Native-Language other than English. If the NLS collating sequence should be used to sort the indices, type Y, otherwise the default N will be used. **Only select NLS if the custom path will contain alphanumeric data.** If it will contain only binary numeric values, use the default of N.

SITEST and SIREPAIR utilities

The SITEST utility is used for checking the integrity of the SUPERDEX B-trees and their correspondence to the data entries they represent. Additionally, SITEST verifies the KWEXCLUDE keyword exclusion SI-path against the KWEXCLUD disk file.

SIREPAIR adds the capability to actually repair the SUPERDEX B-trees, so they will match the data. SIREPAIR will repair 1% of the indexes based on the number of entries in the dataset. There is an upper limit of one million indexes that can be repaired. This is because the process of repairing the indexes is slower than the process of reorganizing the indexes. If more than 1% of the indexes are corrupted, it shows major corruption on the B-tree and therefore the B-tree should be completed reorganized. This can only happen when a maintenance program is executed without the SUPERDEX libraries included during the process.

Inconsistencies are reported by an ASCII/octal/hex dump of the SI-indices in error. Also, each inconsistency indicates whether it is an SI-index with no corresponding entry or an entry with no corresponding SI-index.

Access requirements

Before running SITEST or SIREPAIR, make sure:

- you have shared (DBOPEN *mode* 5) access to the database
- you are logged on as the database creator
- you are logged into the group and account in which the database resides

Because SITEST and SIREPAIR do extensive locking, it is recommended that they not be run during heavy user access.

The rules and processes of SIREPAIR are the same as SITEST. Therefore, first SITEST will be covered, then the differences between SITEST and SIREPAIR.

Invoking SITEST

Two versions of the SITEST program are provided, in PUB.SUPERDEX and NOPRIV.SUPERDEX. They are identical except the **NOPRIV** version does not use techniques that require PM capability. The **PUB** version should be used to obtain the best performance.

Use the **NOPRIV** version if you do not want to run software with PM capability and on datasets whose block sizes are not even multiples of 128 words (128, 256, 384, 512, etc.).

Path Display Screen

```
AUG 28, 1991              SUPERDEX PATH DISPLAY SCREEN              10:53 AM

        Dataset: _____  Selection Order: __
        SI-Path Name: _____
        Path Type: _____
        Key Length: _____   Number of Keys: __   Min Key word length: __
        Index Length: ____            Avg No Key Words: ___
                                     (or Avg No of Indexes for Custom Index)
        Items            Type Length    Items            Type Length

S1 _____  __  ____   S2 _____  __  ____
S3 _____  __  ____   S4 _____  __  ____
   _____  __  ____      _____  __  ____
   _____  __  ____      _____  __  ____
   _____  __  ____      _____  __  ____
   _____  __  ____      _____  __  ____
   _____  __  ____      _____  __  ____
   _____  __  ____      _____  __  ____
   _____  __  ____      _____  __  ____
   _____  __  ____      _____  __  ____
   _____  __  ____      _____  __  ____


SUPERDEX (C) Bradmark Technologies, Inc. 1991
```

The **Path Display Screen** is used to display all related information about a path. SI-Paths that have been added during this execution of SUPERDEX, along with the previously existing SI-Paths can be displayed. Of course, all of the fields on this screen are display only.

Specifying mode of operation

SITEST has two modes of operation:

Mode 1 checks the integrity of the B-tree itself and does not validate its relationship to the data entries.

Mode 2 checks the integrity of the B-tree (same as mode 1) and the correspondence of the SI-indices to the data entries they represent. Mode 2 is slower than mode 1 but performs a more thorough analysis.

```
MODE (1=TREE, 2=FULL) >
```

Mode 1 processing

If mode 1 was selected, SITEST processes each B-tree in succession and displays the phase TREETEST while processing. Once the processing has completed, the number of SI-indices checked is displayed:

```
MODE (1=TREE, 2=FULL) > 1
PROCESSING SI-PATH KWEXCLUDE         OF
TreeTest          0 INDICES          CPU 0:00:00.0 Elapsed 0:00:00
PROCESSING SI-PATH CUSTOMER-NAME     OF CUSTOMERS
TreeTest       1003 INDICES          CPU 0:00:00.3 Elapsed 0:00:01
PROCESSING SI-PATH CUSTOMER-NAME-KW OF CUSTOMERS
TreeTest       3059 INDICES          CPU 0:00:00.4 Elapsed 0:00:01
PROCESSING SI-PATH ADDRESS1-CITY-KW OF CUSTOMERS
TreeTest       4418 INDICES          CPU 0:00:00.5 Elapsed 0:00:01
PROCESSING SI-PATH CUSTOMER-NUMBER   OF ORDER-HEADERS
TreeTest       2620 INDICES          CPU 0:00:00.3 Elapsed 0:00:00
PROCESSING SI-PATH ORDER-TYPE        OF ORDER-HEADERS
TreeTest       2620 INDICES          CPU 0:00:00.2 Elapsed 0:00:00
PROCESSING SI-PATH ORDER-PART        OF ORDER-LINES
TreeTest       9272 INDICES          CPU 0:00:01.7 Elapsed 0:00:03
PROCESSING SI-PATH PART-ORDER        OF ORDER-LINES
TreeTest       9272 INDICES          CPU 0:00:01.7 Elapsed 0:00:03

END OF PROGRAM
```

Item Screen

```
 AUG 28, 1991                SUPERDEX ITEM SCREEN                    10:53 AM


        Dataset: _____  Selection Order: __
        SI-Path Name: _____
        Path Type: _____

    Items in Dataset:


 __ XXXXXXXXXXXXXXXX   __ XXXXXXXXXXXXXXXX   __ XXXXXXXXXXXXXXXX __ XXXXXXXXXXXXXXXX
 __ XXXXXXXXXXXXXXXX   __ XXXXXXXXXXXXXXXX   __ XXXXXXXXXXXXXXXX __ XXXXXXXXXXXXXXXX
 __ XXXXXXXXXXXXXXXX   __ XXXXXXXXXXXXXXXX   __ XXXXXXXXXXXXXXXX __ XXXXXXXXXXXXXXXX
 __ XXXXXXXXXXXXXXXX   __ XXXXXXXXXXXXXXXX   __ XXXXXXXXXXXXXXXX __ XXXXXXXXXXXXXXXX
 __ XXXXXXXXXXXXXXXX   __ XXXXXXXXXXXXXXXX   __ XXXXXXXXXXXXXXXX __ XXXXXXXXXXXXXXXX
 __ XXXXXXXXXXXXXXXX   __ XXXXXXXXXXXXXXXX   __ XXXXXXXXXXXXXXXX __ XXXXXXXXXXXXXXXX
 __ XXXXXXXXXXXXXXXX   __ XXXXXXXXXXXXXXXX   __ XXXXXXXXXXXXXXXX __ XXXXXXXXXXXXXXXX
 __ XXXXXXXXXXXXXXXX   __ XXXXXXXXXXXXXXXX   __ XXXXXXXXXXXXXXXX __ XXXXXXXXXXXXXXXX
 __ XXXXXXXXXXXXXXXX   __ XXXXXXXXXXXXXXXX   __ XXXXXXXXXXXXXXXX __ XXXXXXXXXXXXXXXX
 __ XXXXXXXXXXXXXXXX   __ XXXXXXXXXXXXXXXX   __ XXXXXXXXXXXXXXXX __ XXXXXXXXXXXXXXXX
 __ XXXXXXXXXXXXXXXX   __ XXXXXXXXXXXXXXXX   __ XXXXXXXXXXXXXXXX __ XXXXXXXXXXXXXXXX
 __ XXXXXXXXXXXXXXXX   __ XXXXXXXXXXXXXXXX   __ XXXXXXXXXXXXXXXX __ XXXXXXXXXXXXXXXX


 SUPERDEX (C) Bradmark Technologies, Inc. 1991
```

The **Item Screen** is used to select the IMAGE item(s) that will be included in the SI-Path being built. If the SI-Path already exists, the IMAGE item(s) that were selected previously will be marked appropriately. Otherwise, no items will be mark.

The first four fields are all display only and are passed from the **Path Screen**. The **Dataset:** is the name of the dataset being accessed, the **Selection Order:** is the selection order keyed from the **Dataset Menu**, the **SI-Path Name:** is the name of the SI-Path being accessed, and the **Path Type:** is the description of the type of the SI-Path as defined on the **Path Screen**.

Next, is a table of the IMAGE item names that are contained in the dataset. The first field is used to select the item for inclusion in the SI-Path. The field can be any group of two characters (i.e. **A**, **CX**, **FF**, **B**, etc.). A designation can only be used once per SI-Path.

Numeric designations are not allowed, except when adding a Concatenated SI-Path. In this case, the SI-SUBKEY-1 must be marked with an **A1**, SI-SUBKEY-2 must be marked with an **A2**, SI-SUBKEY-3 with **A3**, and SI-SUBKEY-4 with **A4**.

If the SI-Path was **NOT** marked as grouped on the **Path Screen**, then the items must be marked with **A1, A2, A3, A4** or **A** (if a simple SI-Path).

If a new SI-Path is being created with the same name as an IMAGE item, the IMAGE item will be designated with **A1**, and can not be modified.

When creating a new grouped SI-Path, it is important to designate the IMAGE item you want first in the group as **A** (please refer to *Section 3* for information on the first item in a group).

```
PROCESSING SI-PATH CUSTOMER-NAME-KW OF CUSTOMERS
TreeTest      3062 INDICES              CPU 0:00:00.4 Elapsed 0:00:00
INPUT         1004 RECORDS     100 %    CPU 0:00:02.4 Elapsed 0:00:03
SORT          3077 INDICES              CPU 0:00:00.0 Elapsed 0:00:00
COMPARE       3062 INDICES     100 %    CPU 0:00:01.4 Elapsed 0:00:02
PROCESSING SI-PATH ADDRESS1-CITY-KW OF CUSTOMERS
TreeTest      4418 INDICES              CPU 0:00:00.5 Elapsed 0:00:01
INPUT         1004 RECORDS     100 %    CPU 0:00:03.7 Elapsed 0:00:04
SORT          4449 INDICES              CPU 0:00:00.0 Elapsed 0:00:00
COMPARE       4418 INDICES     100 %    CPU 0:00:02.1 Elapsed 0:00:02
PROCESSING SI-PATH ORDER-TYPE      OF ORDER-HEADERS
TreeTest      2620 INDICES              CPU 0:00:00.2 Elapsed 0:00:00
INPUT         2620 RECORDS     100 %    CPU 0:00:02.2 Elapsed 0:00:02
SORT          2620 INDICES              CPU 0:00:00.0 Elapsed 0:00:00
COMPARE       2620 INDICES     100 %    CPU 0:00:01.1 Elapsed 0:00:01
PROCESSING SI-PATH ORDER-PART      OF ORDER-LINES
TreeTest      9272 INDICES              CPU 0:00:01.5 Elapsed 0:00:02
INPUT         9272 RECORDS     100 %    CPU 0:00:08.6 Elapsed 0:00:09
SORT          9272 INDICES              CPU 0:00:00.0 Elapsed 0:00:00
COMPARE       9272 INDICES     100 %    CPU 0:00:05.4 Elapsed 0:00:06
PROCESSING SI-PATH PART-ORDER      OF ORDER-LINES
TreeTest      9272 INDICES              CPU 0:00:01.6 Elapsed 0:00:02
INPUT         9272 RECORDS     100 %    CPU 0:00:08.9 Elapsed 0:00:10
SORT          9272 INDICES              CPU 0:00:00.0 Elapsed 0:00:00
COMPARE       9272 INDICES     100 %    CPU 0:00:05.8 Elapsed 0:00:06

END OF PROGRAM
```

For each inconsistency detected, the SI-index is displayed with a counter indicating the relative
position in the B-tree. If the counter is on the left side of the output, it indicates an SI-index with no
corresponding entry; if the counter is on the right side, there is an entry with no corresponding SI-
index.

Running SITEST in batch

SITEST can be run in batch, and uses the same dialog as on-line. The method for creating a job
stream by which to run SITEST in batch is to anticipate the on-line prompts and provide responses
for them.

SITEST will QUIT (not TERMINATE) normally upon encountering any error in batch, permitting
testing of the system JCW.

Item Definition Screen

```
AUG 28, 1991          SUPERDEX ITEM DEFINITION SCREEN          10:53 AM

       Dataset: _____  Selection Order: __
       SI-Path Name: _____
       Path Type: _____


       Item: _____   Item Selection Order: ___

       Item length (in bytes):          ___

       Starting Position (in bytes):    ___

       Minimum Number of Characters:    _      (1 - 4)
          per Keyword (Keyword only)

       Average Number of Keywords:      __     (1 - 16)
          per record (Keyword only)

SUPERDEX (C) Bradmark Technologies, Inc. 1991
```

The **Item Definition Screen** is used to define detailed information about the IMAGE items that were selected on the previous screen. This screen will only allow information for fields based the SI-Path type and the IMAGE item type. For example, a numeric item will not be displayed because the IMAGE definition must be used, or if the SI-Path is not a keyword path the fields marked **Keyword only** do not apply.

The first four fields are all display only and are passed from previous screens, the **Dataset:** is the name of the dataset being accessed, the **Selection Order:** is the selection order keyed from the **Dataset Menu**, the **SI-Path Name:** is the name of the SI-Path being accessed, and the **Path Type:** is the description of the type of the SI-Path as defined on the **Path Screen**.

The next two fields, **Item:** and **Item Selection Order:** are also display fields (Passed from the **Item Screen**).

The **Item length:** field is used to enter the length of the SI-Subkey in bytes. By default, for non-keyworded SI-Paths, the length will be set to the length of the IMAGE item. For keyworded SI-Paths, the length will be defaulted to eight (8) bytes.

The **Starting Position:** is used to define the position of where the index should begin. For non-keyworded SI-Paths, the value can be any starting position (relative to 1), as long as the starting position is not longer then the length of the IMAGE item. For keyworded SI-Paths, this value will be forced to one (1).

The next two fields are only valid for keyworded SI-Paths. The first, **Minimum Number of Characters:**, is used to define the minimum number of characters that must be in the word before it will be index. The default is two (2). This means that single character words, such as "a" will not be indexed. The valid values are 1 through 4, inclusive.

SICOUNT utility

SICOUNT was created to provide the exact compression information in the B-trees. SUPERDEX compresses B-trees based on the full index values for a physical SI data record. This means all of the indexes in a given physical SI data record will be compressed the same.

SUPERDEX begins compressing with the left most word and will compress up to the whole length of an SI-INDEX, including the SI-EXTENSION. The default size of the SI-ITEM is set with this compression algorithm taken into consideration. It is possible to increase, or decrease, the compression by manually setting up the SI-ITEM and SI dataset structures (although this is not recommended). Depending on the data and the type of items in the SI-Path, a smaller SI-ITEM can produce a greatly increased compression ratio.

☎ **Since there are several items to consider, contact Bradmark Technical Support before adjusting the structure of the SI-ITEM and SI datasets.**

SICOUNT will not only provide the information on the compression, but will also give the height of the B-trees, the number of leaf and tree records, and the number of indexes in the path.

Invoking SICOUNT

Two versions of the SICOUNT program are provided, in PUB.SUPERDEX and NOPRIV.SUPERDEX. They are identical except the **NOPRIV** version does not use techniques that require PM capability. The **PUB** version should be used to obtain the best performance.

Use the **NOPRIV** version if you do not want to run software with PM capability and on datasets whose block sizes are not even multiples of 128 words (128, 256, 384, 512, etc.).

To invoke SICOUNT:

```
:RUN SICOUNT.PUB.SUPERDEX
```

☞ SICOUNT is run without *;LIB=G* or *;LIB=P*.

Specifying the database

Specify the name of a SUPERDEX'ed database, as shown:

```
DATABASE > OEDB
```

Related Masters Screen

```
Aug 28, 1991          SUPERDEX RELATED MASTERS SCREEN          10:53 AM

    SELECT WITH   AVAILABLE MASTERS | REQUIRED MASTERS   UNQUALIFIED MASTERS
      AN "X"                        |

      __          _____     | _____      _____
      __          _____     | _____      _____
      __          _____     | _____      _____
      __          _____     | _____      _____
      __          _____     | _____      _____
      __          _____     | _____      _____
      __          _____     | _____      _____
      __          _____     | _____      _____
      __          _____     | _____      _____
      __          _____     | _____      _____
      __          _____     | _____      _____
      __          _____     | _____      _____
      __          _____     | _____      _____
      __          _____     | _____      _____
      __          _____     | _____      _____
      __          _____     | _____      _____


 SUPERDEX (C) Bradmark Technologies, Inc. 1991
```

The `Related Master Screen` is used to identify the related Master datasets to be included
in the Supergrouped path being built. This screen has three columns of Master datasets identified.
The AVAILABLE MASTERS column lists those master datasets that can optionally be included in the
current Supergrouped path. The REQUIRED MASTERS column lists those master datasets that
must be included in the current Supergrouped path (e.g. the same path already exists in a particular
master set). The UNQUALIFIED MASTERS column identified those master datasets that cannot be
included in the current Supergrouped path.

Mark any desired optional master dataset(s) to be included in the current Supergrouped path with an
"X". This program will automatically create the necessary paths for each of the master selected
datasets (or modify the existing master path to become a Supergrouped path).

Related Masters Screen Function Keys		
Key	Label	Description
F1	HELP	Displays the Help Screen.
F2		(Not Defined)
F3		(Not Defined)
F4		(Not Defined)
F5		(Not Defined)
F6		(Not Defined)
F7		(Not Defined)
F8	EXIT	Exits back to the Main Menu.

Once all of the data has been keyed, press the ENTER key . The process will then return to the
PATH screen.

Following is a table explaining all of the items displayed:

Label	Description
KeyLength	The total length of the SI-index, including the SI-extension, in words.
BTree Height	The height of the B-tree. This will defined the maximum number of physical I-Os to qualify a record. If the B-tree records are in memory, there will be no physical I-Os.
# Tree Records	The total number of physical records that make up the B-tree.
Contiguous pairs	The number of physical records included in the B-tree, that are contiguous in the SI-dataset (Also called "Index Efficiency"). The percentage of efficiency is also reported.
# Leaf Records	The number of physical records that are located on the bottom of the tree.
# Keys	The total number of SI-indices in the B-tree.
LenPrefix	The number of words in the SI-index that are compressed. "0" is no compression. "3" is three words compressed in every index in the physical record.
#Records	The number of physical records that are compressed at the level.
#Keys	The number of SI-indices that are compressed at the level.
Avg Keys/Rec	The average number of SI-indices that are contained on one physical record.
Max Keys/Rec	The maximum number of SI-indices that are contained on one of the physical records.
Optimal Keys/Rec	The optimal number of SI-indices that can be contained on one physical record.

Execute Screen

```
 AUG 28, 1991              SUPERDEX EXECUTE SCREEN                    10:53 AM


 _   (1)  Execute On-line
     (2)  Execute in Batch
     (3)  Save Job file [Enter file name]: _____
                                            file.group

 _   (1)  Modify Index Structure and Populate Indices
     (2)  Modify Index Structure, only


 Job Name:          _____

 User Name:         _____   Password:   _____

 Group Name:        _____   Password:   _____

 Account Name:  XXXXXXXX  Password:   _____


 SUPERDEX (C) Bradmark Technologies, Inc. 1991
```

This screen is used to select how the modification process should be executed. By default, the process will execute the SUPERDEX maintenance program, SIMAINT, as a son process using the information captured.

The first field is used to select the execution option. **1**, the default, runs the process immediately on-line, **2** streams the process for batch immediately, and **3** is used to save the information entered in a job file that can be streamed at a later time.

If **2** was chosen, a job file named **SIBCHFLE** will be created and streamed. If for some reason SIMAINT does not complete successfully (i.e. the SI dataset did not have enough free space to add the new index), this file will not be automatically purged. Otherwise, it will be purged by the job stream itself. If **3** was chosen, the second field is used to enter a valid MPE file name (with the group optional) to save the job file.

The third field selects whether the process will only define the new SUPERDEX index structure, or will also populate the SUPERDEX indices for the new structure. The default is to populate the indices during the definition execution. To not process the indices, simply enter **2** and at some later point, it may be necessary to reorganize the newly created indices.

The last group of fields are used to define the logon necessary for the job file. The Job Name: is an optional name that defaults to **SIBATCH** and can be modified.

The User Name: and Password: are required fields for logon. The name will default to the current logon USER name and can be modified. The password field is used to enter the user's logon password, if it exists.

SITRACE utility

SITRACE is used to trace all IMAGE intrinsics and SUPERDEX intrinsics called in a program. It is very useful for debugging or logging all data base updates with SUPERDEX.

Activating SITRACE

To active the trace, the JCW SI-TRACE has to be set to a non-zero value before running the program to be traced:

    :SETJCW SITRACE = 1   (or 111 for more detail)

Function

When processing DBOPEN, SUPERDEX checks if the **SITRACE** JCW is set to a non-zero value. If so, an internal flag is set to generate trace information.

☞ **The JCW is checked only by DBOPEN, so it must be set before opening the data base. BREAKing a running program and RESUMEing it after setting the JCW will produce trace information only for data bases opened after the BREAK.**

Redirection of Output

Trace output is sent to the file SITRACEF, which is defaulted to $STDLIST and opened with AccessOptions "share" and "append". This allows you

   a) to trace access via several access paths (multiple DBOPENs to one or more databases),
                              and
   b) to append more trace information to an existing file.

Output can be redirected by specifying a file equation for SITRACEF.

SIPATH utility

SIPATH is a program that will display the IMAGE keys and chains, along with the SUPERDEX SIPATH information. This information is given in a concise and is complete.

To use SIPATH, simply:

```
:RUN SIPATH.PUB.SUPERDEX
```

At this point you will be asked to enter the data base name you wish to look at.

The data base name can be fully qualified with group and account. **You must have correct access to look at data bases outside your logon group or account.** Entering RETURN at the data base name will exit SIPATH.

Type Code	Description
B???	Index blank values
C	Concatenated Index
CG	Concatenated-Grouped Index
CK	Concatenated-Grouped-Keyworded Index
CUS	Custom Index
G	Grouped Index
GK	Grouped-Keyworded Index
IND	Independent Index
K	Keyworded Index
KEX	Keyword Exclude Index
S	Supergrouped Index
SC	Supergrouped-Concatenated Index
SCK	Supergrouped-Concatenated-Keyworded Index
SIM	Simple Index
SK	Supergrouped-Keyworded Index

SIDRIVER utility

SIDRIVER is a utility which permits IMAGE intrinsics to be ``driven'' interactively for various utility functions. It executes an intrinsic and returns the elapsed and CPU times.

The commands are very similar to HP's DBDRIVER utility. There have been some modifications and additions to the standard DBDRIVER utility to support all advanced SUPERDEX retrieval capabilities, new intrinsic parameters and modes, and new intrinsics. To execute:

```
:RUN SIDRIVER.PUB.SUPERDEX
```

☞ Only the differences between SIDRIVER and DBDRIVER are documented.

The modified commands are:

!B	new format is *#{;base}*, where: # is a number 1 to 5 (used to assign multiple databases) *{;base}* is the base name to open (only used during open)
!Q	Always upshifted
!L	Always upshifted
!I	Always upshifted

The new commands are:

!P	Password parameter, if lower case is necessary
/STON	Set a switch to always print the status array after a call
/STOF	Turns status print off
/OCON	Report octal status, in addition to decimal
/OCOF	Turns octal status print off
/HXON	Report hex status, in addition to decimal
/HXOF	Turns hex status print off

New intrinsics are callable by the following commands:

DE	DBERASE
DX	DBDELIX
PX	DBPUTIX

This version of SIDRIVER does not support the **PRIV** entry point and can be excecuted in either compatibility mode or native mode, depending on your machine.

```
------------------
ORDER-HEADERS
  SI-PATH# = 10006 CUSTOMER-NUMBER     SIM    4        8     1
                   - CUSTOMER-NUMBER   I      2
                       (OFFSET =    1   SUBKEY LENGTH =     4)
  SI-PATH# = 10007 ORDER-NUMBER        BSIM   4        4     1
                   - ORDER-NUMBER      I      2
                       (OFFSET =    1   SUBKEY LENGTH =     4)
------------------
ORDER-LINES
  SI-PATH# = 10008 ORDER-PART          C     18       22     2
                   - ORDER-NUMBER      I      2
                       (OFFSET =    1   SUBKEY LENGTH =     4)
                   - PART-NUMBER       X     14
                       (OFFSET =    1   SUBKEY LENGTH =    14)
  SI-PATH# = 10009 PART-ORDER          C     18       22     2
                   - PART-NUMBER       X     14
                       (OFFSET =    1   SUBKEY LENGTH =    14)
                   - ORDER-NUMBER      I      2
                       (OFFSET =    1   SUBKEY LENGTH =     4)
  SI-PATH# = 10010 PART-DESCRIPTION    K      8       12     1     2     4
                   - PART-DESCRIPTION X      26
                       (OFFSET =    1   SUBKEY LENGTH =     8)
------------------
SI

*SPECIAL PATHS*
  SI-PATH# = 10001   KWEXCLUDE         KEX            12


DATABASE> RETURN

END OF PROGRAM
```

Specify the SI dataset(s) Capacities

You will now be asked to enter the desired capacities for each of the SI datasets you want to create.

```
Enter the CAPACITIES for each set separated by comma's OR
enter one CAPACITY followed by @ to be used for ALL sets: 2500@
```

From here, the program will build the new SI-index database and all the requested SI datasets.

```
NUMBER OF ERROR MESSAGES: 0
ROOT FILE OEDBSI CREATED.

The dataset OEDBSI was successfully built.  You may now proceed
with your SUPERDEX installation.
```

This program will also add the necessary SI item and SI dataset to the database being indexed. This is necessary because the SI dataset in the main database instructs SUPERDEX that the indices are being maintained in a separate database.

To invoke SITEST:

```
:RUN SITEST.PUB.SUPERDEX

SITEST VERSION 3.1 (12DEC91) COPYRIGHT DR. MATT/IABG (1991)
```

☞ SITEST is run without **;LIB=G** or **;LIB=P**.

Specifying the database

Specify the name of a SUPERDEX'ed database, as shown:

```
DATABASE > OEDB
```

Specifying datasets

Specify @ to diagnose all datasets, the name of a dataset that contains SI-paths, or **SPACE + RETURN** to diagnose a independent SI-path:

```
DATA SET > @
```

Specifying SI-paths

If a value other than @ was specified for dataset, a prompt is issued to determine whether to diagnose a specific SI-path for the current dataset or all SI-paths for the dataset. Enter an SI-path name or @ to diagnose all SI-paths for the dataset:

```
SI PATH > @
```

Specifying the datasets

Specify the name of the appropriate SI dataset that you want to change the capacity for:

```
SETNAME: SI1
```

If you want to review a list of the SI datasets available for capacity change, enter a "?" in the dataset field and a list of the available SI datasets will be displayed:

```
Enter the name or number of the detail set to have its capacity changed.

Enter the name of number of one of the detail sets in this base:

    #  Dataset          Type      Capacity    Entries      % Full

    1  SI1              Detail    1500        1079         71.9
    2  SI2              Detail    3           0            0

 ----------------------------------------------------------------------------

Enter the dataset NAME or NUMBER:

         SETNAME: SI1
```

The program will now display some supporting information about the dataset identified for capacity change:

```
The new capacity may be specified in any of the following formats :

    Absolute capacity      : number          (e.g. "5501")
    Relative change        : +/- number      (e.g. "+500")
    Percent change         : +/- change %     (e.g. "-25%")
    CTA trending if available for set         (e.g. "C")


Current set capacity      : 1500    (71.9% full)
Current High Water Mark   : 1129
Current set entries       : 1079
Current blocking factor   : 2
Current block size        : 1024 words


Current file size         : 6016 sectors
```

Mode 2 processing

If mode 2 was selected, SITEST processes each B-tree in succession and displays the phase TREETEST while performing the B-tree check. Following the TREETEST, the phases INPUT, SORT, and COMPARE are executed and displayed accordingly. Once the processing of each SI-path is completed, the number of SI-indices checked is displayed:

```
MODE (1=TREE, 2=FULL) > 2
PROCESSING SI-PATH KWEXCLUDE        OF
TreeTest            0 INDICES          CPU 0:00:00.0 Elapsed 0:00:00


PROCESSING SI-PATH CUSTOMER-NAME     OF CUSTOMERS
TreeTest         1004 INDICES          CPU 0:00:00.2 Elapsed 0:00:00
INPUT            1004 RECORDS    100 % CPU 0:00:00.9 Elapsed 0:00:01
SORT             1004 INDICES          CPU 0:00:00.0 Elapsed 0:00:00
COMPARE             0 INDICES


 *** Inconsistency: Keys from SI Chain vs. Keys from Dataset *****


SI Key #   Key Value (SI Chain or Dataset)                DSet Key #

           052105 051524 020055 020116 047440 044516 042105 054040    957
             T  E   S  T    -     N  O    I  N   D  E   X
             54 45  53 54  20 2D  20 4E  4F 20  49 4E  44 45  58 20
           020040 020040 020040 020040 020040 020040 020040 000020
                                                             . . . . . .
             20 20  20 20  20 20  20 20  20 20  20 20  20 20  00 10
           172107
           . . .  G
             F4 47


      957  052105 051524 020055 020116 047440 051105 041517 051104
             T  E   S  T    -     N  O    R  E   C  O   R  D
             54 45  53 54  20 2D  20 4E  4F 20  52 45  43 4F  52 44
           020040 020040 020040 020040 020040 020040 020040 000041
                                                             . . .  !
             20 20  20 20  20 20  20 20  20 20  20 20  20 20  00 21
           164216
           . . . . . .
             E8 8E

 1004 INDICES FROM SI CHAIN, 1004 INDICES FROM DATASET COMPARED
2 ERROR(S) found
```

ALTPROG utility

The ALTPROG utility is used to add capabilities to and increase the MAXDATA of object program files. Native Mode programs on MPE/XL cannot be changed with ALTPROG.

ALTPROG unconditionally:

- adds DS and MR capability
- increases MAXDATA to 32000

Access requirements

Before running ALTPROG, make sure:

- you have write access to the program files
- you are logged into the group and account in which the program files reside

Invoking ALTPROG

To invoke ALTPROG:

```
:RUN ALTPROG.PUB.SUPERDEX

ALTPROG V1.0 (30Jan90)   COPYRIGHT IABG

Changes PROG File(s): Capabilities DS, MR are switched ON,
                MAXDATA is set to 32000.
```

Specifying the program file

Specify the name of an object program file, which may be qualified with group name. The @ wildcard may be specified for either program file name or group name or both, as shown:

```
Expects File Specification in LISTF format

Enter File Specification (RETURN to end program) : @.PROG
```

Invoking SIREPAIR

As stated earlier, while discussing SIREPAIR, we will only identify the differences.

To invoke SIREPAIR:

```
:RUN SIREPAIR.PUB.SUPERDEX;LIB=P

SIREPAIR Version 3.1 (12DEC91) COPYRIGHT DR. MATT/IABG (1991)
```

☞ SIREPAIR MUST be executed with ;**LIB=P**

Specifying Input

The input for database, datasets, and si-paths are the same for SIREPAIR as they are for SITEST.

Specifying request before update

SIREPAIR will now prompt to see if the updating should be done immediately, or should the user specify that the si-path should be repaired.

```
PROMPT BEFORE REPAIR (Y,N) >
```

After this, SIREPAIR will do the same processing as SITEST mode 2. After it displays the indexes that are corrupt, it will prompt the user if the previous question was answered yes.

```
Repair this path ? (Y,N) >
```

If the SI-path should be repaired, SIREPAIR will repair the reported problems and then go on to the next SI-path.

QUERY/3000 utility

QUERY/3000 is compatible with SUPERDEX in that any entries added, deleted, or modified in QUERY while referencing the SUPERDEX SL will automatically adjust the SI-indices, although no advanced retrieval capabilities are available.

To use QUERY to add, delete, and update entries, copy QUERY.PUB.SYS into a group or account in which the SUPERDEX SL segments reside (e.g. new group SUPERDEX.SYS), and then run it to reference the SL:

```
:RUN QUERY.SUPERDEX.SYS;LIB=G
```

It is recommended that QUERY be run to reference the SUPERDEX SL only for adding, updating, and deleting entries and not for retrieving entries.

If using QUERY via the SUPERDEX SL to find entries using a positive value in an item of data type P or Z for which an identically-named SI-path exists, qualifying entries may be returned twice. This is because QUERY internally issues two DBFINDs--one with a signed *argument*, one with an unsigned *argument*--whereas SUPERDEX treats them identically. To avoid this, configure the SI-paths with names different than the item names.

Specifying datasets

Specify @ to diagnose all datasets, the name of a dataset that contains SI-paths, or SPACE + RETURN to diagnose a independent SI-path:

```
DATA SET > @
```

Specifying SI-paths

If a value other than @ was specified for dataset, a prompt is issued to determine whether to diagnose a specific SI-path for the current dataset or all SI-paths for the dataset. Enter an SI-path name or @ to diagnose all SI-paths for the dataset:

```
SI PATH > @
```

Process

At this point SICOUNT will process the SI-Paths selected and will display detailed information about each path:

```
KeyLength        =
BTree Height     =
# Tree Records   =
# Leaf Records   =
# Keys           =

LenPrefix #Records    #Keys Avg.Keys/Rec Max Keys/Rec Optimal Keys/Rec
                                     CPU 0:00:00.0 Elapsed 0:00:00
```

The first is the DATASET prompt:

```
DATASET>
```

Enter the name of the dataset that is located in the database opened by the user program to use against the selection. Press return to exit SuperSELECT.

Next, the SIPATH prompt will be displayed:

```
SIPATH>
```

Enter the SI-Path name to use against the selection. The SI-Path must be one that is located in the dataset specified in the previous prompt. Pressing RETURN will cause SuperSELECT to return to the DATASET prompt.

Finally, the argument will be prompted for:

```
ARGUMENT>
```

Enter any valid SUPERDEX argument, including relational and Boolean operators. Once the argument has been entered the message "Record written", will be displayed for verification to the user.

SuperSELECT will then prompt for more arguments, until RETURN is pressed at the prompt. This allows for multiple search values to be entered. When RETURN is pressed, SuperSELECT will back up to prompt for another SI-Path.

```
:RUN SUPERSEL.PUB.SUPERDEX

SuperSELECT Version 3.1 (12DEC91) Copyright Bradmark Technologies, Inc. (1991)
DATASET> CUSTOMERS
SIPATH> CUSTOMER-NAME-KW
ARGUMENT>
-UNI@ AND CHU@;
Record written
ARGUMENT>
RETURN
SIPATH> RETURN
DATASET> RETURN
End of Program
:RUN USERPROG
```

In the above example, SuperSELECT is used to change the serial search in USERPROG. When **USERPROG** executes, a SUPERDEX selection against the **CUSTOMER-NAME-KW** keyworded SI-

Here is an example of SICOUNT.

```
SICOUNT VERSION 3.1 (19NOV91) COPYRIGHT DR. MATT / IABG (1991)
DATABASE > OEDB
DATA SET > CUSTOMERS

PROCESSING SI-PATH CUSTOMER-NAME    OF CUSTOMERS

KeyLength          =          17
BTree Height       =           3
# Tree Records     =          39
Contiguous pairs   =          37 ( 97.37 %)
# Leaf Records     =          36
# Keys             =        1004


LenPrefix #Records     #Keys  Avg.Keys/Rec  Max Keys/Rec  Optimal Keys/Rec
0              26      679            26            28              29
1              10      290            29            29              30
                                   CPU 0:00:00.4 Elapsed 0:00:01

PROCESSING SI-PATH CUSTOMER-NAME-KW OF CUSTOMERS

KeyLength          =           6
BTree Height       =           2
# Tree Records     =          37
Contiguous pairs   =          35 ( 97.22 %)
# Leaf Records     =          36
# Keys             =        3062


LenPrefix #Records     #Keys  Avg.Keys/Rec  Max Keys/Rec  Optimal Keys/Rec
0              32     2601            81            83              84
1               3      294            98            98              99
4               1      132           132           132             249
                                   CPU 0:00:00.4 Elapsed 0:00:00

PROCESSING SI-PATH ADDRESS1-CITY-KW OF CUSTOMERS

KeyLength          =           6
BTree Height       =           2
# Tree Records     =          49
Contiguous pairs   =          48 (100.00 %)
# Leaf Records     =          48
# Keys             =        4418


LenPrefix #Records     #Keys  Avg.Keys/Rec  Max Keys/Rec  Optimal Keys/Rec
0              41     3353            82            83              84
1               3      288            96            98              99
4               4      730           183           248             249
                                   CPU 0:00:00.4 Elapsed 0:00:00
```

If the SuperSELECT temporary file still exists, SuperSELECT will then display a message and ask if the older file should be purged.

```
Error Closing SuperSELECT file!
  Temporary file already exists.   Purge File (Y/N)?
```

If the older temporary file should be purge, enter **Y**, otherwise enter **N**.

SuperSELECT - Method 3

Method 3 of SuperSELECT is used primarily for a process that will accept the argument values from the user, and then a batch job is run, using the values. This allows a simple program to be written that will prompt the user for the arguments and then write the selection criteria to a flat file. The batch job will always run SuperSELECT, specifying that the flat file should be used.

This is done by running SuperSELECT like:

```
:RUN SUPERSEL.PUB.SUPERDEX;INFO="^^ARGFILE"
```

This tells SuperSELECT to read the dataset, SI-path, and argument from the file *ARGFILE*. The file name can be fully qualified, including the group and account. The format of the *;INFO* string is:

```
:RUN SUPERSEL.PUB.SUPERDEX;INFO="^^filename.group.account"
```

The file layout is similar to the *;INFO* string format used in method 2. It should be a unnumbered flat file, that includes the dataset name, si-path name, and the argument, separated by semi-colon, and no blank lines:

```
dataset;sipath1;argument1;
dataset;sipath2;argument2;
      .
      .
      .
```

For our example, the *ARGFILE* would contain one line:

```
CUSTOMERS;CUSTOMER-NAME-KW;~UNI@ AND CHU@;
```

Examples:

A. Redirect output to another free terminal:

`:FILE SITRACEF;DEV=nn`

B. Append output to an existing trace file:

`:FILE SITRACEF=oldtrace,OLD`

C. Create a new file and redirect output to it:

`:BUILD mytrace;REC=-80,3,F,ASCII;DISC=1000`
`:FILE SITRACEF=mytrace,OLD`

D. Specify file to be created by SUPERDEX:

`:FILE SITRACEF=mytrace,NEW;REC=-80,3,F,ASCII;DISC=1000;SAVE`

☞ Examples A, B, and C allow a trace access via multiple access paths (multiple DBOPENs to one or more databases), whereas D only allows a trace access via the access path of the first DBOPEN.

Since QUERY first issues a DBOPEN with an empty BASE parameter and *mode* 0 (to get TurboIMAGE version information) before prompting, example D is can not be used (only the first DBOPEN will be traced, and nothing else).

The screen file is a file built and designed by the programmer. It simply has a two (2) character command code, followed by a 78 character command or comment:

```
ccxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
  ^-Command or Comment
^-Command Code
```

The valid command codes are:

Code	Command	Description
Spaces	Any comment or blank	Does nothing, only for internal comments.
AA	DATASET;SIPATH;	**ACCEPT-ARGUMENT**: Accepts any valid SuperSELECT argument from the user, and writes out the data.
D	Any string, including escape sequences	**DISPLAY**: Displays the Command portion without a carriage return.
DC	Any string, including escape sequences	**DISPLAY-CARRIAGE RETURN**: Displays the Command portion and then executes a carriage
DP	Any string, including escape sequences	return. **DISPLAY-PROMPT**: Displays the Command portion, then waits for the user to press RETURN. No data is accepted.
R	Any valid program name	**RUN**: Executes the program specified as a son process. No RUN parameters are valid.
S	DATASET;SIPATH;ARGUMENT;	**SET**: Sets a fixed argument value for the dataset and SI-path

SIBASE Utility

The SIBASE utility is provided to allow you to automatically create an SI-index database, which is a separate database used to hold the indexes. This option is mainly chosen for those databases that are enabled for logging. By moving the indexes to a separate database, the log files will not fill up with the accesses to the SI dataset(s).

Access Requirements

Before running SIBASE, make sure:

- No other process is accessing the database
- You are logged on as the CREATOR of the database in the group and account in which the database resides

Invoking SIBASE

To invoke SIBASE:

```
:RUN SIBASE.PUB.SUPERDEX

SIBASE VERSION 3.1 (30DEC91) COPYRIGHT BRADMARK TECHNOGIES, INC (1991)
```

Specifying the database

Specify the name of the SUPERDEX'ed database, as shown:

```
Please enter the PRIMARY data base name, (ENTER to exit): OEDB
```

Specifying the number of SI datasets to build

Specify the number of SI datasets you wish to build. Up to seven (7) different SI datasets can be built if desired.

```
Please enter the NUMBER of SI datasets you wish to build=> 2
```

SISIZE Utility

The SISIZE utility is provided to allow you to change the capacity of an SI (SUPERDEX Index) dataset. The SI set(s) can reside in either the original database being indexed or the external SI-index database. This utility will only function on SI type datasets.

Access Requirements

Before running SISIZE, make sure:

- No other process is accessing the database
- You have a current backup of the database
- You are logged on as the CREATOR of the database in the group and account in which the database resides

Invoking SISIZE

To invoke SISIZE:

```
:RUN SISIZE.PUB.SUPERDEX

SISIZE VERSION 3.1 (30DEC91) COPYRIGHT BRADMARK TECHNOLOGIES, INC. (1991)
```

Specifying the database

Specify the name of the SUPERDEX'ed database, as shown:

```
SOURCE DATABASE: OEDB
       PASSWORD: (N/A)
```

☞ The PASSWORD is not applicable if you are the CREATOR of the database.

Confirmation of the backup

You will be asked to verify that a current backup of the database does exist:

```
Do you have a current backup of the Database ? (Y/N) : Y
```

Retrieving all entries in a set in ascending sorted order

This example retrieves all the entries in the ORDER dataset in ascending sorted sequential order. It uses DBFIND *mode* 100 to position at the alphabetic beginning of the set, and DBGET *mode* 15 to perform a greater-than-or-equal-to retrieval.

```
WORKING-STORAGE SECTION.
01 ORDER.
       05 CUST-NUMBER       PIC 9(6)
       05 ORDER-NUMBER      PIC 9(6).
       05 ORDER-DATE        PIC 9(6).
       05 ARTICLE-NUMBER    PIC 9(10).
       05 AMOUNT            PIC 9(11) COMP-3.

01 BASE                     PIC X(8)  VALUE " OEDB ".
01 DSET                     PIC X(16) VALUE "ORDER           ".
01 ITEM                     PIC X(16) VALUE "ORDER-DATE      ".
01 LIST                     PIC XX    VALUE "@".

01 STAT.
       05 CONDITION-WORD    PIC S9(4) COMP.
       05 STAT2             PIC S9(4) COMP.
       05 STAT3-4           PIC S9(9) COMP.
       05 STAT5-6           PIC S9(9) COMP.
       05 STAT7-8           PIC S9(9) COMP.
       05 STAT9-10          PIC S9(9) COMP.

01 MODE15                   PIC S9(4) COMP VALUE 15.
01 MODE100                  PIC S9(4) COMP VALUE 100.
01 DUMMY                    PIC X.
```

Specifying the new capacity:

Specify the desired capacity for the SI dataset as follows:

```
Enter new capacity        : 2500
```

The program will now display some additional supporting information:

```
Proposed capacity         : 2500    (43.2% full)

Proposed file size        : 10016 sectors
Change in file size       : +4000 sectors
```

The program will ask for verification of the new capacity:

```
Is the new capacity of 2500 correct ? (Y/N) : Y
```

The program will now execute the requested capacity change while displaying the following information:

```
Capacity change in progress --

   1079 entries copied

Detail set change successfully completed
```

At this point, the program will prompt for another dataset. You may enter the name of another SI dataset or press **RETURN** to exit the program.

Retrieving all entries in a set in descending sorted order

This example retrieves all the entries in the ORDER dataset in descending sorted sequential order. It uses DBFIND mode 200 to position at the alphabetic beginning of the set, and DBGET mode 16 to perform a less-than-or-equal-to retrieval.

```
WORKING-STORAGE SECTION.
01 ORDER.
      05 CUST-NUMBER      PIC 9(6)
      05 ORDER-NUMBER     PIC 9(6).
      05 ORDER-DATE       PIC 9(6).
      05 ARTICLE-NUMBER   PIC 9(10).
      05 AMOUNT           PIC 9(11) COMP-3.

01 BASE                   PIC X(8)  VALUE "  OEDB  ".
01 DSET                   PIC X(16) VALUE "ORDER           ".
01 ITEM                   PIC X(16) VALUE "ORDER-DATE      ".
01 LIST                   PIC XX    VALUE "@".

01 STAT.
      05 CONDITION-WORD   PIC S9(4) COMP.
      05 STAT2            PIC S9(4) COMP.
      05 STAT3-4          PIC S9(9) COMP.
      05 STAT5-6          PIC S9(9) COMP.
      05 STAT7-8          PIC S9(9) COMP.
      05 STAT9-10         PIC S9(9) COMP.

01 MODE16                 PIC S9(4) COMP VALUE 16.
01 MODE200                PIC S9(4) COMP VALUE 200.
01 DUMMY                  PIC X.
```

ALTPROG will optionally verify for each program file whether to change or not. Respond **Y** to be prompted for each program file or **N** to change all program files without verification:

```
Verify before changing file ? (Y/N) N

Processing Account AR, Group PROG

Processing File ARADDHS.PROG.AR
Processing File ARADDRP.PROG.AR
WARNING: Native Mode Program ARCHGRP.PROG.AR not changed.
Processing File ARREPSM.PROG.AR
Processing File ARREPDM.PROG.AR
**** File ARREPDM.PROG.AR could not be opened :
EXCLUSIVE VIOLATION: FILE BEING ACCESSED  (FSERR 90)
Processing File ARREPTT.PROG.AR
. . .
```

As shown above, ALTPROG displays an error message for each program file that it cannot change.

Once all program files in the specified fileset are changed, ALTPROG prompts for a new file specification. Enter it, or hit RETURN to exit the program:

```
Enter File Specification (RETURN to end program) : RETURN

END OF PROGRAM
```

Running ALTPROG in batch

ALTPROG cannot be run in batch.

Retrieving entries using a partial or generic key

This example illustrates a partial-key retrieval against the CUST dataset, in which the SI-path *CUST-NAME* is accessed. The user enters a value, and the program appends an @ to perform partial-key retrieval.

```
WORKING-STORAGE SECTION.
01 CUST.
     05 CUST-NUMBER      PIC 9(6).
     05 CUST-NAME        PIC X(20).
     05 CUST-ADDRESS1    PIC X(20).
     05 CUST-ADDRESS2    PIC X(20).

01 SEARCH-NAME           PIC X(20).

01 BASE                  PIC X(8)  VALUE "  OEDB  ".
01 DSET                  PIC X(16) VALUE "CUST            ".
01 ITEM                  PIC X(16) VALUE "CUST-NAME       ".
01 LIST                  PIC XX    VALUE "@".
01 ARGUMENT              PIC X(20).

01 STAT.
     05 CONDITION-WORD   PIC S9(4) COMP.
     05 STAT2            PIC S9(4) COMP.
     05 STAT3-4          PIC S9(9) COMP.
     05 STAT5-6          PIC S9(9) COMP.
     05 STAT7-8          PIC S9(9) COMP.
     05 STAT9-10         PIC S9(9) COMP.

01 MODE1                 PIC S9(4) COMP VALUE 1.
01 MODE5                 PIC S9(4) COMP VALUE 5.
01 DUMMY                 PIC X.
```

SuperSELECT was create to provide a simple means of changing serial reads to SUPERDEX reads with no program changes to existing software. This is especially helpful when using third-party software, or when source code is not available.

SuperSELECT works by interrupting a serial read being executed and replacing it with a SUPERDEX read. It can be executed using one of four methods.

The only restriction is that the multi-database relational access is not available. This is because SuperSELECT does not open or process any data. It sets some switches and builds a temporary file of the selection information. When the user program executes, the serial read will be intercepted and the data stored in the temporary file will be used to replace the serial read with a SUPERDEX read.

☞ **Within any of the four methods of execution, it is important to remember that only datasets and SI-Paths that exists in the database that the user program serially reads are valid.**

Invoking SuperSELECT

There are no special requirements to invoke SuperSELECT. Simply execute SuperSELECT immediately prior to the normal execution of the user program.

SuperSELECT - Method 1

The first method for executing SuperSELECT is for testing and on-the-fly selections. SuperSELECT will loop through the prompts, allowing multiple arguments for a single SI-Path, multiple SI-Paths for a single dataset, and multiple datasets for a single database. This allows for full relational access within the database of the user program.

Run the SuperSELECT program:

```
:RUN SUPERSEL.PUB.SUPERDEX

SuperSELECT Version 3.1(12DEC91)Copyright Bradmark Technologies, Inc.
```

At this point SuperSELECT will loop through three prompts.

Retrieving entries using a concatenated key

This example shows a lookup against the concatenated SI-key comprised of the items ORD-DAT (X6) and ARTICLE-NUMBER (X10). The DBFIND mode -116 represents the combined length, in bytes, of the two fields.

```
WORKING-STORAGE SECTION.
01 ORDER.
      05 CUST-NUMBER       PIC 9(6)
      05 ORDER-NUMBER      PIC 9(6).
      05 ORDER-DATE        PIC 9(6).
      05 ARTICLE-NUMBER    PIC 9(10).
      05 AMOUNT            PIC 9(11) COMP-3.

01 BASE                    PIC X(8)  VALUE "  OEDB  ".
01 DSET                    PIC X(16) VALUE "ORDER            ".
01 ITEM                    PIC X(16) VALUE "ORDER-DATE       ".
01 LIST                    PIC XX    VALUE "@".
01 ARGUMENT.
      05 SEARCH-DATE       PIC 9(6).
      05 SEARCH-ARTICLE    PIC 9(10).

01 STAT.
      05 CONDITION-WORD    PIC S9(4) COMP.
      05 STAT2             PIC S9(4) COMP.
      05 STAT3-4           PIC S9(9) COMP.
      05 STAT5-6           PIC S9(9) COMP.
      05 STAT7-8           PIC S9(9) COMP.
      05 STAT9-10          PIC S9(9) COMP.

01 SI-MODE                 PIC S9(4) COMP VALUE -116.
01 MODE5                   PIC S9(4) COMP VALUE 5.
01 DUMMY                   PIC X.
```

Path in the **CUSTOMERS** dataset will be done. The records selected will be the records that contain a word starting with **UNI** and a word starting with **CHU**.

If SuperSELECT is executed twice, before any serial read program, the temporary file will still exist. SuperSELECT will then display a message and ask if the older file should be purged.

```
Error Closing SuperSELECT file!
   Temporary file already exists.  Purge File (Y/N)?
```

If the older temporary file should be purge, enter **Y**, otherwise enter **N**.

SuperSELECT - Method 2

The second method for running SuperSELECT is used primarily prior to a batch run of a program, where there will only be one argument entered and that argument is known prior to when the batch job executes.

On the RUN command, the **;INFO** string is used to pass the dataset name, SI-Path name, and the single argument value. For example:

```
:RUN SUPERSEL.PUB.SUPERDEX;INFO="CUSTOMERS;CUSTOMER-NAME-KW;-UNI@ AND CHU@;"

SuperSELECT Version 3.1(12DEC91)Copyright Bradmark Technologies, Inc.
DATASET>
CUSTOMERS
SIPATH>
CUSTOMER-NAME-KW
ARGUMENT>
~UNI@ AND CHU@;
Record written
```

This run command will set the same selection criteria as the example for method 1. The format of the **;INFO** string is:

```
:RUN SUPERSEL.PUB.SUPERDEX;INFO="dataset; si-path; argument; "
```

The rules for *dataset, si-path*, and *argument* are the same as described in the introduction of SuperSELECT.

Retrieving entries in a range of values

This example locates all the entries in the range between START-DATE and END-DATE. The >= and <= operators are embedded with the values entered to form the *argument*.

```
WORKING-STORAGE SECTION.
01 ORDER.
    05 CUST-NUMBER      PIC 9(6)
    05 ORDER-NUMBER     PIC 9(6).
    05 ORDER-DATE       PIC 9(6).
    05 ARTICLE-NUMBER   PIC 9(10).
    05 AMOUNT           PIC 9(11) COMP-3.

01 BASE               PIC X(8)  VALUE "  OEDB  ".
01 DSET               PIC X(16) VALUE "ORDER           ".
01 ITEM               PIC X(16) VALUE "ORDER-DATE      ".
01 LIST               PIC XX    VALUE "@".
01 ARGUMENT.
    05 FILLER           PIC XX    VALUE ">=".
    05 START-DATE       PIC 9(6).
    05 FILLER           PIC XX    VALUE "<=".
    05 END-DATE         PIC 9(6).
    05 FILLER           PIC X     VALUE "@".

01 STAT.
    05 CONDITION-WORD   PIC S9(4) COMP.
    05 STAT2            PIC S9(4) COMP.
    05 STAT3-4          PIC S9(9) COMP.
    05 STAT5-6          PIC S9(9) COMP.
    05 STAT7-8          PIC S9(9) COMP.
    05 STAT9-10         PIC S9(9) COMP.

01 MODE1              PIC S9(4) COMP VALUE 1.
01 MODE5              PIC S9(4) COMP VALUE 5.
01 DUMMY              PIC X.
```

When SuperSELECT runs it will display the values as it processes:

```
:RUN SUPERSEL.PUB.SUPERDEX;INFO=""^ARGFILE"
SuperSELECT Version 3.1(12DEC91)Copyright Bradmark Technologies, Inc.
DATASET>
CUSTOMERS
SIPATH>
CUSTOMER-NAME-KW
ARGUMENT>
~UNI@ AND CHU@;
Record written
```

If the SuperSELECT temporary file still exists, SuperSELECT will then display a message and ask if the older file should be purged.

```
Error Closing SuperSELECT file!
  Temporary file already exists.  Purge File (Y/N)?
```

If the older temporary file should be purge, enter **Y**, otherwise enter **N.**

SuperSELECT - Method 4

Method 4 is a very powerful and user-friendly way to execute SuperSELECT on-line. It is used to allow a user to run SuperSELECT, input the argument(s), and automatically run the user's program, all through a data-entry screen.

This allows for great flexibility and automation for SuperSELECT, without having to train the user on how SuperSELECT works. Screens can be customized to match the appearance that the user expects, which will allow SuperSELECT to become an integral part of the user's application.

Before SuperSELECT is executed, enter a file equation for **SSSCREEN**, which can be fully qualified. Then run SuperSELECT without the **;INFO** string. SuperSELECT will open and process the customized screen.

```
:FILE SSSCREEN=CUSTOM.group.account
:RUN SUPERSEL.PUB.SUPERDEX
```

Retrieving entries using multiple values

This example illustrates a search using multiple values OR'ed together. The user is prompted for names of cities in a loop, with one city entered per prompt, and presses RETURN when all cities have been specified. The program strings together the cities entered, delimits them with square brackets and embeds | operators to form a single *argument* for DBFIND.

```
WORKING-STORAGE SECTION.
01 CUST.
     05 CUST-NUMBER      PIC 9(6).
     05 CUST-NAME        PIC X(20).
     05 CUST-ADDRESS1    PIC X(20).
     05 CUST-ADDRESS2    PIC X(20).

01 CITY                  PIC X(20).

01 BASE                  PIC X(8)  VALUE "  OEDB  ".
01 DSET                  PIC X(16) VALUE "CUST            ".
01 ITEM                  PIC X(16) VALUE "CUST-OFFICE     ".
01 LIST                  PIC XX    VALUE "@".
01 ARGUMENT              PIC X(200).

01 STAT.
     05 CONDITION-WORD   PIC S9(4) COMP.
     05 STAT2            PIC S9(4) COMP.
     05 STAT3-4          PIC S9(9) COMP.
     05 STAT5-6          PIC S9(9) COMP.
     05 STAT7-8          PIC S9(9) COMP.
     05 STAT9-10         PIC S9(9) COMP.

01 MODE1                 PIC S9(4) COMP VALUE 1.
01 MODE5                 PIC S9(4) COMP VALUE 5.
01 DUMMY                 PIC X.
01 ITERATION             PIC S9(4) COMP VALUE 0.
```

Following is an example (the Command Codes are in bold for documentation purpose):

```
    This is an example of a SuperSELECT Screen file.
    All of the commands codes are used in this example.  Of
    course, they can be used in any order, and they are not
    case sensitive.
DC(escape home and clear)
DC                      SuperSELECT Example
DC
DC    This is an example of how SuperSELECT screen files should be
DC    built.  This example will prompt for a part description
DC    keyword for dataset ORDER-LINES and SI-Path PART-DESC.
DC    It will also force an argument of "S" (shipped) for the
DC    SI-Path SHIPPED-FLAG.
DC
DC          First, is the prompt for PART-DESC.
DC
    [notice we do not carriage return now]
D   Please enter the Part Description argument:
AAORDER-LINES;PART-DESC;
DC
          Next, we force the shipped status.
S ORDER-LINES;SHIPPED-FLAG;S;

DP   Press RETURN to continue:
    Now start the user's program
R USERPROG
```

Now, the screen display of how to execute SuperSELECT with method 4 and what is displayed:

```
:FILE SSSCREEN=USERSCRN
:RUN SUPERSEL.PUB.SUPERDEX
(screen is homed and cleared)
                     SuperSELECT Example

    This is an example of how SuperSELECT screen files should be
    built.  This example will prompt for a part description
    keyword for dataset ORDER-LINES and SI-Path PART-DESC.
    It will also force an argument of "S" (shipped) for the
    SI-Path SHIPPED-FLAG.


          First, is the prompt for PART-DESC.

 Please enter the Part Description argument: -RED AND BEAR@;

    Press RETURN to continue: RETURN
```

At this point the user's program (USERPROG) from the **R** command would be executed. The program may display the report on the screen, or actually write out a print file.

Refining and undoing a selection

The following example illustrates a generalized search routine in which the user is prompted in a loop for full or partial keywords contained in part descriptions and whether to AND, OR, or AND NOT each pair, undo the last selection, or display the qualifying entries, start a new search, or exit. After each keyword is entered, DBFIND is called and the number of qualifying entries is displayed, so the user can decide whether to continue to refine the selection; if too few entries qualify, the user can undo the last selection and impose an alternate keyword.

```
WORKING-STORAGE SECTION.

01 BASE              PIC X(8)    VALUE "  OEDB  ".
01 DSET              PIC X(16)   VALUE "PART            ".
01 ITEM              PIC X(16)   VALUE "PART-DESCRIPTION".
01 LIST              PIC XX      VALUE "@".
01 ARGUMENT          PIC X(260)  VALUE SPACE.

01 STAT.
    05 CONDITION-WORD PIC S9(4)  COMP.
    05 STAT2          PIC S9(4)  COMP.
    05 STAT3-4        PIC S9(9)  COMP.
    05 STAT5-6        PIC S9(9)  COMP.
    05 STAT7-8        PIC S9(9)  COMP.
    05 STAT9-10       PIC S9(9)  COMP.

01 MODE1             PIC S9(4)   COMP VALUE 1.
01 MODE5             PIC S9(4)   COMP VALUE 5.
01 SEARCH-STRING     PIC X(20).
01 WHAT-TO-DO        PIC X.
01 RELOP             PIC XX SPACE.
01 QUALIFY           PIC Z(8)9.

01 PART-ID.
    05 PART-NUMBER    PIC X(10).
    05 PART-DESCRIP   PIC X(50).
```

COBOL Program examples

Examples of COBOL programs that perform various types of SUPERDEX access appear on the following pages, as shown below:

- ■ **Retrieving all entries in a set in ascending sorted order**
- ■ **Retrieving all entries in a set in descending sorted order**
- ■ **Retrieving entries using a partial or generic key**
- ■ **Retrieving entries using a concatenated key**
- ■ **Retrieving entries in a range of values**
- ■ **Retrieving entries using multiple values**
- ■ **Refining and undoing a selection**
- ■ **Retrieving entries using multiple SI-paths in a single dataset**
- ■ **Retrieving entries using multiple datasets**
- ■ **Retrieving entries using multiple databases**
- ■ **Retrieving entries in multiple sets and bases using projection**
- ■ **Reading SI-indices only**
- ■ **Customizing SI-key value(s) with SIUSER**
- ■ **Manually adding SI-indices with DBPUTIX**
- ■ **Manually deleting SI-indices with DBDELIX**
- ■ **Adding SI-dataset to the lock descriptor**
- ■ **Calling SITRANSLATE**

```
PROMPT-STRING.
   DISPLAY "Enter part description keyword:".
   ACCEPT SEARCH-STRING.
   STRING ARGUMENT DELIMITED BY SPACE
         "[ "        DELIMITED BY SIZE
         SEARCH-STRING
                     DELIMITED BY SPACE
         "@]"        DELIMITED BY SIZE
         RELOP       DELIMITED BY SPACE
               INTO ARGUMENT.

   CALL "DBFIND" USING BASE, DSET, MODE1, STAT, ITEM, ARGUMENT.
   IF CONDITION-WORD <> 0 AND <> 17
      GO TO DB-ERROR.

   MOVE STAT5-6 TO QUALIFY.
   DISPLAY QUALIFY "entries qualify.".
   MOVE "/" TO ARGUMENT.
   MOVE "  " TO RELOP.

   GO TO PROMPT-WHAT-TO-DO.

DISPLAY-PART-ID.
   CALL "DBGET" USING BASE, DSET, MODE5, STAT, LIST, PART-ID, DUMMY
   IF CONDITION-WORD <> 0
      IF CONDITION-WORD = 15
         DISPLAY "End of Selection"
         MOVE SPACES TO ARGUMENT
         GO TO PROMPT-STRING
      ELSE
         GO TO DB-ERROR.
   DISPLAY PART-NUMBER "  " PART-DESCRIP.
   GO TO DISPLAY-PART-ID.
```

```
PROCEDURE DIVISION.

MAIN.
    PERFORM OPEN-BASE.

    CALL "DBFIND" USING BASE, DSET, MODE100, STAT, ITEM, DUMMY.

    IF CONDITION-WORD <> 0
       IF CONDITION-WORD = 17
          DISPLAY "no entry" STOP RUN
       ELSE
          GO TO DB-ERROR.

    DISPLAY "Cust#   Order#  Date    Article    Amount".
    PERFORM GET-NEXT UNTIL CONDITION-WORD = 11.
    STOP RUN.

GET-NEXT SECTION.
    CALL "DBGET" USING BASE, DSET, MODE15, STAT, LIST, ORDER, DUMMY.
    IF CONDITION-WORD <> 0 AND <> 11 GO TO DB-ERROR.
    DISPLAY CUST-NUMBER " " ORDER-NUMBER " " ORDER-DATE " "
        ARTICLE-NUMBER " " AMOUNT.
GET-NEXT-EXIT.
    EXIT.
```

```
PROCEDURE DIVISION.

MAIN.
    PERFORM OPEN-BASE.

    DISPLAY "Enter customer number:".
    ACCEPT SEARCH-CUST.
    DISPLAY "Enter first order date:".
    ACCEPT START-DATE.
    DISPLAY "Enter last order date:".
    ACCEPT END-DATE.

    CALL "DBFIND" USING BASE, DSET, MODE1, STAT, ITEM1, ARGUMENT1.
    IF CONDITION-WORD <> 0
       IF CONDITION-WORD = 17
          DISPLAY "no entry"
          STOP RUN
       ELSE
          GO TO DB-ERROR.

    CALL "DBFIND" USING BASE, DSET, MODE1, STAT, ITEM2, ARGUMENT2.
    IF CONDITION-WORD <> 0
       IF CONDITION-WORD = 17
          DISPLAY "no entry"
          STOP RUN
       ELSE
          GO TO DB-ERROR.

    DISPLAY "Cust#   Order#  Date     Article     Amount".
    PERFORM GET-NEXT STAT5-6 TIMES.
    STOP RUN.

GET-NEXT SECTION.
    CALL "DBGET" USING BASE, DSET, MODE5, STAT, LIST, ORDER, DUMMY.
    IF CONDITION-WORD <> 0
       GO TO DB-ERROR.
    DISPLAY CUST-NUMBER "  " ORDER-NUMBER "  " ORDER-DATE "  "
         ARTICLE-NUMBER "  " AMOUNT.

GET-NEXT-EXIT.
    EXIT.
```

```
PROCEDURE DIVISION.

MAIN.
    PERFORM OPEN-BASE.

    CALL "DBFIND" USING BASE, DSET, MODE200, STAT, ITEM, DUMMY.

    IF CONDITION-WORD <> 0
        IF CONDITION-WORD = 17
            DISPLAY "no entry"
            STOP RUN
        ELSE
            GO TO DB-ERROR.

    DISPLAY "Cust#   Order#  Date     Article      Amount".
    PERFORM GET-NEXT UNTIL CONDITION-WORD = 10.
    STOP RUN.


GET-NEXT SECTION.
    CALL "DBGET" USING BASE, DSET, MODE16, STAT, LIST, ORDER, DUMMY.
    IF CONDITION-WORD <> 0 AND <> 10 GO TO DB-ERROR.
    DISPLAY CUST-NUMBER "  " ORDER-NUMBER "  " ORDER-DATE "  "
        ARTICLE-NUMBER "  " AMOUNT.

GET-NEXT-EXIT.
    EXIT.
```

```
PROCEDURE DIVISION.

MAIN.
    PERFORM OPEN-BASE.

    DISPLAY "Enter customer name:".
    ACCEPT SEARCH-NAME.
    STRING "[*           DELIMITED BY SIZE
           SEARCH-NAME   DELIMITED BY SPACE
           "@]"          DELIMITED BY SIZE
                INTO ARGUMENT1.
    DISPLAY "Enter first order date:".

    ACCEPT START-DATE.
    DISPLAY "Enter last order date:".
    ACCEPT END-DATE.

    CALL "DBFIND" USING BASE, DSET1, MODE1, STAT, ITEM1, ARGUMENT1.

    IF CONDITION-WORD <> 0
       IF CONDITION-WORD =17
          DISPLAY "no entry"
          STOP RUN
       ELSE
          GO TO DB-ERROR.

    CALL "DBFIND" USING BASE, DSET2, MODE1, STAT, ITEM2, ARGUMENT2.
    IF CONDITION-WORD <> 0
       IF CONDITION-WORD = 17
          DISPLAY "no entry"
          STOP RUN
       ELSE
          GO TO DB-ERROR.

    DISPLAY "Cust#   Name                    Order#  Date"
            "   Article ".
    PERFORM GET-CUST UNTIL CONDITION-WORD <> 0.

GET-CUST SECTION.
    CALL "DBGET" USING BASE, DSET1, MODE5, STAT, LIST, CUST, DUMMY.
    IF CONDITION-WORD <> 0
       GO TO GET-CUST-EXIT.
    CALL "DBFIND" USING BASE, DSET2, MODE1, STAT, NULL-ITEM,
                    CUST-NUM OF CUST.
    IF CONDITION-WORD <> 0
       GO TO DB-ERROR.
    PERFORM GET-ORDER STAT5-6 TIMES.

GET-CUST-EXIT.
    EXIT.
```

```
PROCEDURE DIVISION.

MAIN.
    PERFORM OPEN-BASE.

    DISPLAY "Enter customer name:".
    ACCEPT SEARCH-NAME.
    STRING
        SEARCH-NAME DELIMITED BY SPACE
                "@" DELIMITED BY SPACE
            INTO ARGUMENT.

    CALL "DBFIND" USING BASE, DSET, MODE1, STAT, ITEM, ARGUMENT.
    IF CONDITION-WORD <> 0
        IF CONDITION-WORD =17
            DISPLAY "no entry"
            STOP RUN
        ELSE
            GO TO DB-ERROR.

    DISPLAY "Cust#        Name              Address".
    PERFORM GET-NEXT STAT5-6 TIMES.
    STOP RUN.

GET-NEXT SECTION.
    CALL "DBGET" USING BASE, DSET, MODE5, STAT, LIST, CUST, DUMMY.
    IF CONDITION-WORD <> 0
        GO TO DB-ERROR.
    DISPLAY CUST-NUMBER "  " CUST-NAME "  " CUST-ADDRESS1
        "  " CUST-ADDRESS2.

GET-NEXT-EXIT.
    EXIT.
```

Retrieving entries using multiple databases

This example illustrates a relational retrieval between two databases. The user is prompted for a customer name and a starting and ending date for the customer's orders. A partial-key DBFIND is performed to locate the customer, and the HISTORY-FLAG on the customer record is checked to see if additional orders exist in the OEHIST history database. If so, these orders are also looked up with DBFIND (using the | operator to perform an OR operation, and the & operator to relate the two DBFIND calls), and all related orders are returned and displayed.

```
WORKING-STORAGE SECTION.
01 ORDER.
     05 CUST-NUM         PIC 9(6).
     05 ORDER-NUM        PIC 9(6).
     05 ORDER-DATE       PIC 9(6).
     05 ARTICLE-NUM      PIC 9(10).
     05 AMOUNT           PIC 9(11) COMP-3.
01 CUST.
     05 CUST-NUM         PIC 9(6).
     05 CUST-NAME        PIC X(20).
     05 CUST-ADDRESS1    PIC X(20).
     05 CUST-ADDRESS2    PIC X(20).
     05 HISTORY-FLAG     PIC X.
01 SEARCH-NAME          PIC X(20).
01 BASE1               PIC X(8)  VALUE "  OEDB  ".
01 BASE2               PIC X(8)  VALUE "  OEHIST".
01 DSET1               PIC X(16) VALUE "CUST            ".
01 DSET2               PIC X(16) VALUE "ORDER           ".
01 ITEM1               PIC X(16) VALUE "CUST-NAME       ".
01 ITEM2               PIC X(22) VALUE "ORDER-DATE,CUST-NUM".
01 NULL-ITEM           PIC X     VALUE ";"
01 LIST                PIC XX    VALUE "@".
01 ARGUMENT1           PIC X(24).
01 ARGUMENT2.
     05 FILLER          PIC XXX   VALUE "[>=".
     05 START-DATE      PIC 9(6).
     05 FILLER          PIC XX    VALUE "<=".
     05 END-DATE        PIC 9(6).
     05 FILLER          PIC X     VALUE "]".
     05 RELOP           PIC XXX.
01 STAT.
     05 CONDITION-WORD  PIC S9(4) COMP.
     05 STAT2           PIC S9(4) COMP.
     05 STAT3-4         PIC S9(9) COMP.
     05 STAT5-6         PIC S9(9) COMP.
     05 STAT7-8         PIC S9(9) COMP.
     05 STAT9-10        PIC S9(9) COMP.
01 MODE1               PIC S9(4) COMP VALUE 1.
01 MODE5               PIC S9(4) COMP VALUE 5.
01 DUMMY               PIC X.
01 ID-SAVE             PIC S9(4) COMP.
01 BASE                PIC X(8).
```

```
PROCEDURE DIVISION.

MAIN.
    PERFORM OPEN-BASE.

    DISPLAY "Enter order date:".
    ACCEPT SEARCH-DATE.
    DISPLAY "Enter article number:".
    ACCEPT SEARCH-ARTICLE.

    CALL "DBFIND" USING BASE, DSET, SI-MODE, STAT, ITEM, ARGUMENT.
    IF CONDITION-WORD <> 0
       IF CONDITION-WORD = 17
          DISPLAY "no entry"
          STOP RUN
       ELSE
          GO TO DB-ERROR.

    DISPLAY "Cust#  Order# Date    Article    Amount".
    PERFORM GET-NEXT UNTIL CONDITION-WORD = 15.
    STOP RUN.

GET-NEXT SECTION.
    CALL "DBGET" USING BASE, DSET, MODE5, STAT, LIST, ORDER, DUMMY.
    IF CONDITION-WORD <> 0 AND <> 15
       GO TO DB-ERROR.
    DISPLAY CUST-NUMBER "  " ORDER-NUMBER "  " ORDER-DATE "  "
       ARTICLE-NUMBER "  " AMOUNT.

GET-NEXT-EXIT.
    EXIT.
```

```
GET-CUST SECTION.
    CALL "DBGET" USING BASE, DSET1, MODE5, STAT, LIST, CUST, DUMMY.
    IF CONDITION-WORD <> 0
        GO TO GET-CUST-EXIT.
    CALL "DBFIND" USING BASE1, DSET2, MODE1, STAT, NULL-ITEM,
                        CUST-NUM OF CUST.
    IF CONDITION-WORD <> 0 AND <> 17
        GO TO DB-ERROR.
    MOVE BASE1 TO BASE.
    PERFORM GET-ORDER STAT5-6 TIMES.
    CALL "DBFIND" USING BASE2, DSET2, MODE1, STAT, NULL-ITEM, CUST-NUM OF CUST.
    IF CONDITION-WORD <> 0 AND <> 17
        GO TO DB-ERROR.
    MOVE BASE2 TO BASE.
    PERFORM GET-ORDER STAT5-6 TIMES.

GET-CUST-EXIT.
    EXIT.

GET-ORDER SECTION.
    CALL "DBGET" USING BASE, DSET2, MODE5, STAT, LIST, ORDER, DUMMY.
    IF CONDITION-WORD <> 0
        GO TO DB-ERROR.
    DISPLAY CUST-NUM OF CUST " " CUST-NAME " "
     ORDER-NUM " " ORDER-DATE " " ARTICLE-NUM.

GET-ORDER-EXIT.
    EXIT.
```

```
PROCEDURE DIVISION.

MAIN.
   PERFORM OPEN-BASE.

   DISPLAY "Enter first order date:".
   ACCEPT START-DATE.
   DISPLAY "Enter last order date:".
   ACCEPT END-DATE.

   CALL "DBFIND" USING BASE, DSET, MODE1, STAT, ITEM, ARGUMENT.
   IF CONDITION-WORD <> 0
      IF CONDITION-WORD = 17
         DISPLAY "no entry"
         STOP RUN
      ELSE
         GO TO DB-ERROR.

   DISPLAY "Cust#   Order#  Date    Article    Amount".
   PERFORM GET-NEXT STAT5-6 TIMES.
   STOP RUN.

GET-NEXT SECTION.
   CALL "DBGET" USING BASE, DSET, MODE5, STAT, LIST, ORDER, DUMMY.
   IF CONDITION-WORD <> 0
      GO TO DB-ERROR.
   DISPLAY CUST-NUMBER "  " ORDER-NUMBER "  " ORDER-DATE "  "
      ARTICLE-NUMBER "  " AMOUNT.

GET-NEXT-EXIT.
   EXIT.
```

```
CALL "DBFIND" USING BASE, DSET1, MODE1, ITEM1, ARGUMENT.
IF CONDITION-WORD <> 0
   IF CONDITION-WORD = 17
      DISPLAY "Entry not found"
      STOP RUN
   ELSE
      GO TO DB-ERROR.


MOVE "[*]" TO ARGUMENT.
CALL "DBFIND" USING BASE, DSET2, MODE1, ITEM2, ARGUMENT.
IF CONDITION-WORD <> 0
   IF CONDITION-WORD = 17
      DISPLAY "No corresponding entry in QUOTE-MASTER"
      STOP RUN
   ELSE
      GO TO DB-ERROR.


DISPLAY "Enter start date:".
ACCEPT SEARCH-DATE-START.
DISPLAY "Enter end date:".
ACCEPT SEARCH-END-DATE.
STRING "[>="             DELIMITED BY SIZE
       SEARCH-DATE-START  DELIMITED BY SPACE
       "<="             DELIMITED BY SIZE
       SEARCH-DATE-END   DELIMITED BY SPACE
       "]&"             DELIMITED BY SIZE
            INTO ARGUMENT.


CALL "DBFIND" USING BASE, DSET2, MODE1, ITEM3, ARGUMENT.
IF CONDITION-WORD <> 0
   IF CONDITION-WORD = 17
      DISPLAY "No entry in specified date range"
      STOP RUN
   ELSE
      GO TO DB-ERROR.


DISPLAY "Enter coverage type:".
ACCEPT SEARCH-TYPE.
STRING "["              DELIMITED BY SIZE
       SEARCH-TYPE       DELIMITED BY SPACE
       "]&"             DELIMITED BY SIZE
               INTO ARGUMENT.


CALL "DBFIND" USING BASE, DSET3, MODE1, ITEM4, ARGUMENT.
IF CONDITION-WORD <> 0
   IF CONDITION-WORD = 17
      DISPLAY "No entry with this coverage type"
      STOP RUN
   ELSE
      GO TO DB-ERROR.
```

```
PROCEDURE DIVISION.

MAIN.
    PERFORM OPEN-BASE.

GET-CITIES.
    DISPLAY "Enter city:".
    ACCEPT CITY.
    IF CITY <> SPACE
        IF ITERATION = 0
            STRING "["       DELIMITED BY SIZE
                    CITY      DELIMITED BY SPACE
                    "]"       DELIMITED BY SIZE
                        INTO ARGUMENT
            MOVE 1 TO ITERATION
            GO TO GET-CITIES
        ELSE
            STRING ARGUMENT DELIMITED BY SPACE
                    "["         DELIMITED BY SIZE
                    CITY        DELIMITED BY SPACE
                    "]|" DELIMITED BY SPACE
                        INTO ARGUMENT
            GO TO GET-CITIES.

    CALL "DBFIND" USING BASE, DSET, MODE1, STAT, ITEM, ARGUMENT.
    IF CONDITION-WORD <> 0
        IF CONDITION-WORD = 17
            DISPLAY "no entry"
            STOP RUN
        ELSE
            GO TO DB-ERROR.

    DISPLAY "Cust#        Name                Address".
    PERFORM GET-NEXT STAT5-6 TIMES.
    STOP RUN.

GET-NEXT SECTION.
    CALL "DBGET" USING BASE, DSET, MODE5, STAT, LIST, CUST, DUMMY.
    IF CONDITION-WORD <> 0
        GO TO DB-ERROR.
    DISPLAY CUST-NUMBER "  " CUST-NAME "  " CUST-ADDRESS1
        "  " CUST-ADDRESS2.

GET-NEXT-EXIT.
    EXIT.
```

Reading SI-indices only

In this example, the concatenated SI-key for the ORDER dataset is comprised of the CUST-NUMBER and AMOUNT. By using the ! list, only the SI-indices are returned; because the AMOUNT is part of the SI-key, sufficient information is available in the SI-index without having to read the corresponding data entries.

```
WORKING-STORAGE SECTION.
01 SI-INDEX
    05 CUST-NUMBER      PIC 9(6).
    05 AMOUNT           PIC 9(11) COMP-3.
    05 FILLER           PIC 9(9) COMP.
01 TOTAL                PIC 9(11) COMP-3.
01 TOTAL-OUT            PIC Z(8)9.99.

01 BASE                 PIC X(8)  VALUE "  OEDB  ".
01 DSET                 PIC X(16) VALUE "ORDER           ".
01 ITEM                 PIC X(16) VALUE "CUST-NUMBER     ".
01 LIST                 PIC XX    VALUE "!;".
01 ARGUMENT             PIC 9(6).

01 STAT.
    05 CONDITION-WORD   PIC S9(4) COMP.
    05 STAT2            PIC S9(4) COMP.
    05 STAT3-4          PIC S9(9) COMP.
    05 STAT5-6          PIC S9(9) COMP.
    05 STAT7-8          PIC S9(9) COMP.
    05 STAT9-10         PIC S9(9) COMP.

01 SI-MODE              PIC S9(4) COMP VALUE -106.
01 MODE5                PIC S9(4) COMP VALUE 5.
01 DUMMY                PIC X.
```

```
PROCEDURE DIVISION.

MAIN.
    PERFORM OPEN-BASE.
    GO TO PROMPT-STRING.

PROMPT-WHAT-TO-DO.
    DISPLAY "Enter mode: [A]nd, [O]r, and [N]ot, [U]ndo,".
    DISPLAY "   [D]isplay, new [S]earch, [E]xit:"
    ACCEPT WHAT-TO-DO.
    IF WHAT-TO-DO = "E"
        STOP RUN.
    IF WHAT-TO-DO = "S"
        MOVE SPACE TO ARGUMENT
                      RELOP
        GO TO PROMPT-STRING.
    IF WHAT-TO-DO = "U"
        MOVE "\/" TO ARGUMENT
        MOVE SPACE TO RELOP
        GO TO PROMPT-WHAT-TO-DO.
    IF WHAT-TO-DO = "A"
        MOVE "&" TO RELOP
        GO TO PROMPT-STRING.
    IF WHAT-TO-DO = "O"
        MOVE "|" TO RELOP
        GO TO PROMPT-STRING.
    IF WHAT-TO-DO = "N"
        MOVE "!&" TO RELOP
        GO TO PROMPT-STRING.
    IF WHAT-TO-DO = "D"
        PERFORM DISPLAY-PART-ID
        GO TO PROMPT-WHAT-TO-DO.

    GO TO PROMPT-WHAT-TO-DO.
```

Customizing SI-key value(s) with SIUSER

This example shows an SIUSER procedure that builds a custom SI-key in the ORDER-DETAIL dataset consisting of the second half of the ARTICLE-NUMBER (digits 5 through 10) and the ORDER-DATE for the ARTICLE-DATE SI-path. For the ASCII-DATE si-path it converts the passed double integer date (LAST-UPDATE) in the CUSTOMERS dataset to ASCII so generic and partial lookups can be done.

```
IDENTIFICATION DIVISION.
PROGRAM-ID. SIUSER.
ENVIRONMENT DIVISION.
CONFIGURATION SECTION.

DATA DIVISION.
WORKING-STORAGE SECTION
01 ORDER.
      05 CUST-NUMBER       PIC 9(6).
      05 ORDER-NUMBER      PIC 9(6).
      05 ORDER-DATE        PIC 9(6).
      05 ARTICLE-NUMBER.
          10 FILLER            PIC 9999.
          10 ARTICLE-CODE      PIC 9(6).
      05 AMOUNT            PIC 9(11) COMP-3.


01 CUSTOMERS.
      05 CUST-NUMBER          PIC 9(6).
      05 CUST-NAME            PIC X(30).
      05 CUST-ADDRESS         PIC X(50).
      05 LAST-UPDATE          PIC S9(9) COMP.


LINKAGE SECTION.
01 BASE            PIC X(8).
01 DSET            PIC X(16).
01 ITEM            PIC X(16).
01 BUFFER          PIC X(2000).


01 INDEX-BUF.
      05 INDEX-COUNT       PIC S9(4) COMP.
      05 IB-INDEX          PIC X(50).
      05 IB-ARTICLE-DATE REDEFINES IB-INDEX.
          10 INDEX-ART-CODE   PIC 9(6).
          10 INDEX-DATE       PIC 9(6).
      05 IB-ASCII-DATE REDEFINES IB-INDEX.
          10 INDEX-AD-YY      PIC 99.
          10 INDEX-AD-MM      PIC 99.
          10 INDEX-AD-DD      PIC 99.
```

Retrieving entries using multiple SI-paths in a single dataset

This example shows a relational access retrieval involving two SI-paths in a single dataset. Two DBFIND calls are performed in succession, with square brackets imposed and the & boolean operator appended to the *argument* in the second DBFIND.

```
WORKING-STORAGE SECTION.
01 ORDER.
      05 CUST-NUMBER        PIC 9(6).
      05 ORDER-NUMBER       PIC 9(6).
      05 ORDER-DATE         PIC 9(6).
      05 ARTICLE-NUMBER     PIC 9(10).
      05 AMOUNT             PIC 9(11) COMP-3.

01 BASE                     PIC X(8)   VALUE "  OEDB  ".
01 DSET                     PIC X(16)  VALUE "ORDER          ".
01 ITEM1                    PIC X(16)  VALUE "CUST-NUMBER    ".
01 ITEM2                    PIC X(16)  VALUE "ORDER-DATE     ".
01 LIST                     PIC XX     VALUE "@".
01 ARGUMENT1.
      05 FILLER             PIC X      VALUE "[".
      05 SEARCH-CUST        PIC 9(6).
      05 FILLER             PIC XX     VALUE "] ".
01 ARGUMENT2.
      05 FILLER             PIC XXX    VALUE "[>=".
      05 START-DATE         PIC 9(6).
      05 FILLER             PIC XX     VALUE "<=".
      05 END-DATE           PIC 9(6).
      05 FILLER             PIC XXX    VALUE "]& ".

01 STAT.
      05 CONDITION-WORD     PIC S9(4)  COMP.
      05 STAT2              PIC S9(4)  COMP.
      05 STAT3-4            PIC S9(9)  COMP.
      05 STAT5-6            PIC S9(9)  COMP.
      05 STAT7-8            PIC S9(9)  COMP.
      05 STAT9-10           PIC S9(9)  COMP.

01 MODE1                    PIC S9(4)  COMP VALUE 1.
01 MODE5                    PIC S9(4)  COMP VALUE 5.
01 DUMMY                    PIC X.
```

```
PROCEDURE DIVISION.

MAIN.
   PERFORM OPEN-BASE.

   DISPLAY "Enter customer number:".
   ACCEPT SEARCH-CUST.
   DISPLAY "Enter first order date:".
   ACCEPT START-DATE.
   DISPLAY "Enter last order date:".
   ACCEPT END-DATE.

   CALL "DBFIND" USING BASE, DSET, MODE1, STAT, ITEM1, ARGUMENT1.
   IF CONDITION-WORD <> 0
      IF CONDITION-WORD = 17
         DISPLAY "no entry"
         STOP RUN
      ELSE
         GO TO DB-ERROR.

   CALL "DBFIND" USING BASE, DSET, MODE1, STAT, ITEM2, ARGUMENT2.
   IF CONDITION-WORD <> 0
      IF CONDITION-WORD = 17
         DISPLAY "no entry"
         STOP RUN
      ELSE
         GO TO DB-ERROR.

   DISPLAY "Cust#  Order#  Date    Article    Amount".
   PERFORM GET-NEXT STAT5-6 TIMES.
   STOP RUN.

GET-NEXT SECTION.
   CALL "DBGET" USING BASE, DSET, MODE5, STAT, LIST, ORDER, DUMMY.
   IF CONDITION-WORD <> 0
      GO TO DB-ERROR.
   DISPLAY CUST-NUMBER "  " ORDER-NUMBER "  " ORDER-DATE "  "
         ARTICLE-NUMBER "  " AMOUNT.

GET-NEXT-EXIT.
   EXIT.
```

Retrieving entries using multiple datasets

This example performs a relational access retrieval against the datasets CUST and ORDER which both contain the common item CUST-NUM. Two DBFIND calls are performed in succession with varying parameters, and DBGETs are performed against both datasets to retrieve all the qualifying entries.

```
WORKING-STORAGE SECTION.
01 ORDER.
    05 CUST-NUM         PIC 9(6).
    05 ORDER-NUM        PIC 9(6).
    05 ORDER-DATE       PIC 9(6).
    05 ARTICLE-NUM      PIC 9(10).
    05 AMOUNT           PIC 9(11) COMP-3.
01 CUST.
    05 CUST-NUM         PIC 9(6).
    05 CUST-NAME        PIC X(20).
    05 CUST-ADDRESS1    PIC X(20).
    05 CUST-ADDRESS2    PIC X(20).

01 SEARCH-NAME          PIC X(20).

01 BASE                 PIC X(8)  VALUE "  OEDB  ".
01 DSET1                PIC X(16) VALUE "CUST            ".
01 DSET2                PIC X(16) VALUE "ORDER           ".
01 ITEM1                PIC X(16) VALUE "CUST-NAME       ".
01 ITEM2                PIC X(22) VALUE "ORDER-DATE,CUST-NUM".
01 NULL-ITEM            PIC X     VALUE ";"
01 LIST                 PIC XX    VALUE "@".
01 ARGUMENT1            PIC X(24).
01 ARGUMENT2.
    05 FILLER           PIC XXX   VALUE "[>=".
    05 START-DATE       PIC 9(6).
    05 FILLER           PIC XX    VALUE "<=".
    05 END-DATE         PIC 9(6).
    05 FILLER           PIC XXX   VALUE "]& ".

01 STAT.
    05 CONDITION-WORD   PIC S9(4) COMP.
    05 STAT2            PIC S9(4) COMP.
    05 STAT3-4          PIC S9(9) COMP.
    05 STAT5-6          PIC S9(9) COMP.
    05 STAT7-8          PIC S9(9) COMP.
    05 STAT9-10         PIC S9(9) COMP.
01 MODE1                PIC S9(4) COMP VALUE 1.
01 MODE5                PIC S9(4) COMP VALUE 5.
01 DUMMY                PIC X.
```

```
PROCEDURE DIVISION.

MAIN.
   PERFORM OPEN-BASE.

   DISPLAY "Enter customer name:".
   ACCEPT SEARCH-NAME.
   STRING "[*              DELIMITED BY SIZE
          SEARCH-NAME   DELIMITED BY SPACE
          "@]"           DELIMITED BY SIZE
               INTO ARGUMENT1.
   DISPLAY "Enter first order date:".

   ACCEPT START-DATE.
   DISPLAY "Enter last order date:".
   ACCEPT END-DATE.

   CALL "DBFIND" USING BASE, DSET1, MODE1, STAT, ITEM1, ARGUMENT1.

   IF CONDITION-WORD <> 0
      IF CONDITION-WORD =17
         DISPLAY "no entry"
         STOP RUN
      ELSE
         GO TO DB-ERROR.

   CALL "DBFIND" USING BASE, DSET2, MODE1, STAT, ITEM2, ARGUMENT2.
   IF CONDITION-WORD <> 0
      IF CONDITION-WORD = 17
         DISPLAY "no entry"
         STOP RUN
      ELSE
         GO TO DB-ERROR.

   DISPLAY "Cust#   Name                      Order#  Date"
          "    Article ".
   PERFORM GET-CUST UNTIL CONDITION-WORD <> 0.

GET-CUST SECTION.
   CALL "DBGET" USING BASE, DSET1, MODE5, STAT, LIST, CUST, DUMMY.
   IF CONDITION-WORD <> 0
      GO TO GET-CUST-EXIT.
   CALL "DBFIND" USING BASE, DSET2, MODE1, STAT, NULL-ITEM,
                   CUST-NUM OF CUST.
   IF CONDITION-WORD <> 0
      GO TO DB-ERROR.
   PERFORM GET-ORDER STAT5-6 TIMES.

GET-CUST-EXIT.
   EXIT.
```

```
GET-ORDER SECTION.
    CALL "DBGET" USING BASE, DSET2, MODE5, STAT, LIST, ORDER, DUMMY.
    IF CONDITION-WORD <> 0
        GO TO DB-ERROR.
    DISPLAY CUST-NUM OF CUST "  " CUST-NAME "  "
     ORDER-NUM "  " ORDER-DATE "  " ARTICLE-NUM.

GET-ORDER-EXIT.
    EXIT.
```

Retrieving entries using multiple databases

This example illustrates a relational retrieval between two databases. The user is prompted for a customer name and a starting and ending date for the customer's orders. A partial-key DBFIND is performed to locate the customer, and the HISTORY-FLAG on the customer record is checked to see if additional orders exist in the OEHIST history database. If so, these orders are also looked up with DBFIND (using the | operator to perform an OR operation, and the & operator to relate the two DBFIND calls), and all related orders are returned and displayed.

```
WORKING-STORAGE SECTION.
01 ORDER.
       05 CUST-NUM         PIC 9(6).
       05 ORDER-NUM        PIC 9(6).
       05 ORDER-DATE       PIC 9(6).
       05 ARTICLE-NUM      PIC 9(10).
       05 AMOUNT           PIC 9(11) COMP-3.
01 CUST.
       05 CUST-NUM         PIC 9(6).
       05 CUST-NAME        PIC X(20).
       05 CUST-ADDRESS1    PIC X(20).
       05 CUST-ADDRESS2    PIC X(20).
       05 HISTORY-FLAG     PIC X.
01 SEARCH-NAME            PIC X(20).
01 BASE1                  PIC X(8)  VALUE "  OEDB  ".
01 BASE2                  PIC X(8)  VALUE "  OEHIST".
01 DSET1                  PIC X(16) VALUE "CUST            ".
01 DSET2                  PIC X(16) VALUE "ORDER           ".
01 ITEM1                  PIC X(16) VALUE "CUST-NAME       ".
01 ITEM2                  PIC X(22) VALUE "ORDER-DATE,CUST-NUM".
01 NULL-ITEM              PIC X     VALUE ";"
01 LIST                   PIC XX    VALUE "@".
01 ARGUMENT1              PIC X(24).
01 ARGUMENT2.
       05 FILLER           PIC XXX   VALUE "[>=".
       05 START-DATE       PIC 9(6).
       05 FILLER           PIC XX    VALUE "<=".
       05 END-DATE         PIC 9(6).
       05 FILLER           PIC X     VALUE "]".
       05 RELOP            PIC XXX.
01 STAT.
       05 CONDITION-WORD   PIC S9(4) COMP.
       05 STAT2            PIC S9(4) COMP.
       05 STAT3-4          PIC S9(9) COMP.
       05 STAT5-6          PIC S9(9) COMP.
       05 STAT7-8          PIC S9(9) COMP.
       05 STAT9-10         PIC S9(9) COMP.
01 MODE1                  PIC S9(4) COMP VALUE 1.
01 MODE5                  PIC S9(4) COMP VALUE 5.
01 DUMMY                  PIC X.
01 ID-SAVE                PIC S9(4) COMP.
01 BASE                   PIC X(8).
```

```
PROCEDURE DIVISION.

MAIN.
   PERFORM OPEN-BASE1.

   DISPLAY "Enter customer name:".
   ACCEPT SEARCH-NAME.
   STRING "[ "                DELIMITED BY SIZE
          SEARCH-NAME         DELIMITED BY SPACE
          "@]"               DELIMITED BY SIZE
                   INTO ARGUMENT1.
   DISPLAY "Enter first order date:".
   ACCEPT START-DATE.
   DISPLAY "Enter last order date:".
   ACCEPT END-DATE.
   CALL "DBFIND" USING BASE1, DSET1, MODE1, STAT, ITEM1, ARGUMENT1.
   IF CONDITION-WORD <> 0
      IF CONDITION-WORD = 17
         DISPLAY "no entry"
         STOP RUN
      ELSE
         GO TO DB-ERROR.


   IF HISTORY-FLAG = SPACE
      MOVE "& " TO RELOP
   ELSE
      MOVE SPACE TO RELOP.


   CALL "DBFIND" USING BASE1, DSET2, MODE1, STAT, ITEM2, ARGUMENT2.
   IF CONDITION-WORD <> 17 AND <> 0
      GO TO DB-ERROR.


   IF HISTORY-FLAG <> SPACE
      MOVE STAT2 TO ID-SAVE
      PERFORM OPEN-BASE2
      MOVE ID-SAVE TO STAT2
      MOVE "|& " TO RELOP
      CALL "DBFIND" USING BASE2, DSET2, MODE1, STAT, ITEM2, ARGUMENT2
      IF CONDITION-WORD <> 17 AND <> 0
         GO TO DB-ERROR.


   IF STAT5-6 = 0
      DISPLAY "no entry"
      STOP RUN.


   DISPLAY "Cust#   Name                    Order#  Date    Article ".
   PERFORM GET-CUST UNTIL CONDITION-WORD <> 0.
   STOP RUN.
```

```
GET-CUST SECTION.
    CALL "DBGET" USING BASE, DSET1, MODE5, STAT, LIST, CUST, DUMMY.
    IF CONDITION-WORD <> 0
        GO TO GET-CUST-EXIT.
    CALL "DBFIND" USING BASE1, DSET2, MODE1, STAT, NULL-ITEM,
                        CUST-NUM OF CUST.
    IF CONDITION-WORD <> 0 AND <> 17
        GO TO DB-ERROR.
    MOVE BASE1 TO BASE.
    PERFORM GET-ORDER STAT5-6 TIMES.
    CALL "DBFIND" USING BASE2, DSET2, MODE1, STAT, NULL-ITEM, CUST-NUM OF CUST.
    IF CONDITION-WORD <> 0 AND <> 17
        GO TO DB-ERROR.
    MOVE BASE2 TO BASE.
    PERFORM GET-ORDER STAT5-6 TIMES.

GET-CUST-EXIT.
    EXIT.

GET-ORDER SECTION.
    CALL "DBGET" USING BASE, DSET2, MODE5, STAT, LIST, ORDER, DUMMY.
    IF CONDITION-WORD <> 0
        GO TO DB-ERROR.
    DISPLAY CUST-NUM OF CUST "  " CUST-NAME "  "
     ORDER-NUM "  " ORDER-DATE "  " ARTICLE-NUM.

GET-ORDER-EXIT.
    EXIT.
```

Retrieving entries in multiple sets and bases using projection

This example locates all the quotations for earthquake insurance coverage given to policyholders in Los Angeles in November and December of 1987. This requires four DBFIND calls against three datasets, with the second DBFIND performing the projection.

```
WORKING-STORAGE SECTION.
01 BASE              PIC X(8)  VALUE "  INSURE  ".
01 DSET1             PIC X(16) VALUE "INSURED-MASTER    ".
01 DSET2             PIC X(16) VALUE "QUOTE-MASTER      ".
01 DSET3             PIC X(16) VALUE "QUOTE-DETAIL      ".
01 ARGUMENT          PIC X(26).


01 INSURED-MASTER.
   05 POLICY-NUM     PIC 9(8).
   05 CITY           PIC X(20).

01 QUOTE-MASTER.
   05 QUOTE-NUM      PIC 9(6).
   05 QUOTE-DATE     PIC 9(6).
   05 POLICY-NUM     PIC 9(8).

01 QUOTE-DETAIL.
   05 QUOTE-NUM      PIC 9(6).
   05 COVERAGE-TYPE  PIC XX.

01 ITEM1             PIC X(32) VALUE "CITY,POLICY-NUM".
01 ITEM2             PIC X(32) VALUE "POLICY-NUM,QUOTE-NUM".
01 ITEM3             PIC X(32) VALUE "QUOTE-DATE,QUOTE-NUM".
01 ITEM4             PIC X(32) VALUE "COVERAGE-TYPE,QUOTE-NUM".

01 SEARCH-CITY       PIC X(20).
01 SEARCH-DATE-START PIC 9(6).
01 SEARCH-DATE-END   PIC 9(6).
01 SEARCH-TYPE       PIC XX.

01 MODE1             PIC S9(4) COMP VALUE 1.
01 MODE5             PIC S9(4) COMP VALUE 5.

PROCEDURE DIVISION.

MAIN.
   DISPLAY "Enter city:".
   ACCEPT SEARCH-CITY.
   STRING "["             DELIMITED BY SIZE
          SEARCH-CITY     DELIMITED BY SPACE
          "]"             DELIMITED BY SIZE
             INTO ARGUMENT.
```

```
    CALL "DBFIND" USING BASE, DSET1, MODE1, ITEM1, ARGUMENT.
    IF CONDITION-WORD <> 0
       IF CONDITION-WORD = 17
          DISPLAY "Entry not found"
          STOP RUN
       ELSE
          GO TO DB-ERROR.


    MOVE "[*]" TO ARGUMENT.
    CALL "DBFIND" USING BASE, DSET2, MODE1, ITEM2, ARGUMENT.
    IF CONDITION-WORD <> 0
       IF CONDITION-WORD = 17
          DISPLAY "No corresponding entry in QUOTE-MASTER"
          STOP RUN
       ELSE
          GO TO DB-ERROR.


    DISPLAY "Enter start date:".
    ACCEPT SEARCH-DATE-START.
    DISPLAY "Enter end date:".
    ACCEPT SEARCH-END-DATE.
    STRING "[>="               DELIMITED BY SIZE
           SEARCH-DATE-START   DELIMITED BY SPACE
           "<="                DELIMITED BY SIZE
           SEARCH-DATE-END     DELIMITED BY SPACE
           "]&"                DELIMITED BY SIZE
                INTO ARGUMENT.


    CALL "DBFIND" USING BASE, DSET2, MODE1, ITEM3, ARGUMENT.
    IF CONDITION-WORD <> 0
       IF CONDITION-WORD = 17
          DISPLAY "No entry in specified date range"
          STOP RUN
       ELSE
          GO TO DB-ERROR.


    DISPLAY "Enter coverage type:".
    ACCEPT SEARCH-TYPE.
    STRING "["                 DELIMITED BY SIZE
           SEARCH-TYPE         DELIMITED BY SPACE
           "]&"                DELIMITED BY SIZE
                   INTO ARGUMENT.


    CALL "DBFIND" USING BASE, DSET3, MODE1, ITEM4, ARGUMENT.
    IF CONDITION-WORD <> 0
       IF CONDITION-WORD = 17
          DISPLAY "No entry with this coverage type"
          STOP RUN
       ELSE
          GO TO DB-ERROR.
```

```
GET-ORDER.
    CALL "DBGET" USING BASE, DSET3, MODE5, STAT, LIST, QUOTE-DETAIL, DUMMY.
    IF CONDITION-WORD <> 0
        IF CONDITION-WORD = 15
            DISPLAY "End of selection"
            STOP RUN
        ELSE
            GO TO DB-ERROR.

    DISPLAY QUOTE-NUM OF QUOTE-DETAIL "   " COVERAGE-TYPE.
    GO TO GET-ORDER.
```

Reading SI-indices only

In this example, the concatenated SI-key for the ORDER dataset is comprised of the CUST-NUMBER and AMOUNT. By using the ! list, only the SI-indices are returned; because the AMOUNT is part of the SI-key, sufficient information is available in the SI-index without having to read the corresponding data entries.

```
WORKING-STORAGE SECTION.
01 SI-INDEX
      05 CUST-NUMBER       PIC 9(6).
      05 AMOUNT            PIC 9(11) COMP-3.
      05 FILLER            PIC 9(9) COMP.
01 TOTAL                   PIC 9(11) COMP-3.
01 TOTAL-OUT               PIC Z(8)9.99.

01 BASE                    PIC X(8)   VALUE "  OEDB  ".
01 DSET                    PIC X(16)  VALUE "ORDER           ".
01 ITEM                    PIC X(16)  VALUE "CUST-NUMBER     ".
01 LIST                    PIC XX     VALUE "!;".
01 ARGUMENT                PIC 9(6).

01 STAT.
      05 CONDITION-WORD    PIC S9(4) COMP.
      05 STAT2             PIC S9(4) COMP.
      05 STAT3-4           PIC S9(9) COMP.
      05 STAT5-6           PIC S9(9) COMP.
      05 STAT7-8           PIC S9(9) COMP.
      05 STAT9-10          PIC S9(9) COMP.

01 SI-MODE                 PIC S9(4) COMP VALUE -106.
01 MODE5                   PIC S9(4) COMP VALUE 5.
01 DUMMY                   PIC X.
```

```
PROCEDURE DIVISION.

MAIN.
    PERFORM OPEN-BASE.

    DISPLAY "Enter customer number:".
    ACCEPT CUST-NUMBER.
    MOVE CUST-NUMBER TO ARGUMENT.
    CALL "DBFIND" USING BASE, DSET, SI-MODE, STAT, ITEM, ARGUMENT.
    IF CONDITION-WORD <> 0
        IF CONDITION-WORD = 17
            DISPLAY "no entry"
            STOP RUN
        ELSE
            GO TO DB-ERROR.

    PERFORM GET-NEXT UNTIL CONDITION-WORD = 15.
    MOVE TOTAL TO TOTAL-OUT.
    DISPLAY "Total for customer: " TOTAL-OUT.
    STOP RUN.

GET-NEXT SECTION.
    CALL "DBGET" USING BASE, DSET, MODE5, STAT, LIST, SI-INDEX, DUMMY.
    IF CONDITION-WORD <> 0 AND <> 15
        GO TO DB-ERROR.
    ADD AMOUNT TO TOTAL.

GET-NEXT-EXIT.
    EXIT.
```

Customizing SI-key value(s) with SIUSER

This example shows an SIUSER procedure that builds a custom SI-key in the ORDER-DETAIL dataset consisting of the second half of the ARTICLE-NUMBER (digits 5 through 10) and the ORDER-DATE for the ARTICLE-DATE SI-path. For the ASCII-DATE si-path it converts the passed double integer date (LAST-UPDATE) in the CUSTOMERS dataset to ASCII so generic and partial lookups can be done.

```
IDENTIFICATION DIVISION.
PROGRAM-ID. SIUSER.
ENVIRONMENT DIVISION.
CONFIGURATION SECTION.

DATA DIVISION.
WORKING-STORAGE SECTION
01 ORDER.
    05 CUST-NUMBER      PIC 9(6).
    05 ORDER-NUMBER     PIC 9(6).
    05 ORDER-DATE       PIC 9(6).
    05 ARTICLE-NUMBER.
        10 FILLER          PIC 9999.
        10 ARTICLE-CODE    PIC 9(6).
    05 AMOUNT           PIC 9(11) COMP-3.

01 CUSTOMERS.
    05 CUST-NUMBER          PIC 9(6).
    05 CUST-NAME            PIC X(30).
    05 CUST-ADDRESS         PIC X(50).
    05 LAST-UPDATE          PIC S9(9) COMP.

LINKAGE SECTION.
01 BASE             PIC X(8).
01 DSET             PIC X(16).
01 ITEM             PIC X(16).
01 BUFFER           PIC X(2000).

01 INDEX-BUF.
    05 INDEX-COUNT      PIC S9(4) COMP.
    05 IB-INDEX         PIC X(50).
    05 IB-ARTICLE-DATE REDEFINES IB-INDEX.
        10 INDEX-ART-CODE  PIC 9(6).
        10 INDEX-DATE       PIC 9(6).
    05 IB-ASCII-DATE REDEFINES IB-INDEX.
        10 INDEX-AD-YY      PIC 99.
        10 INDEX-AD-MM      PIC 99.
        10 INDEX-AD-DD      PIC 99.
```

```
PROCEDURE DIVISION USING BASE, DSET, ITEM, BUFFER, INDEX-BUF.

USER.

    IF DSET = "ORDER-DETAIL" AND
       ITEM = "ARTICLE-DATE"
          MOVE BUFFER TO ORDER
          MOVE 1 TO INDEX-COUNT
          MOVE SPACES TO IB-INDEX
          MOVE ARTICLE-CODE TO INDEX-ART-CODE
          MOVE ORDER-DATE TO INDEX-DATE
          GOBACK.

    IF DSET = "CUSTOMERS" AND
       ITEM = "ASCII-DATE"
          MOVE BUFFER TO CUSTOMERS
          MOVE 1 TO INDEX-COUNT
          MOVE SPACES TO IB-INDEX
          COMPUTE INDEX-AD-YY = LAST-UPDATE / 1000000
          COMPUTE INDEX-AD-MM = (LAST-UPDATE - INDEX-AD-YY) / 10000
          COMPUTE INDEX-AD-DD = LAST-UPDATE - INDEX-AD-YY - INDEX-AD-DD
          GOBACK.

*   ITEM and DSET did not match any of our custom SI-Paths.
    GOBACK.
```

Manually adding SI-indices with DBPUTIX

In this example, the user is prompted for information about a new customer, including the customer's initials, if any. The customer is DBPUT into the CUST dataset, and if any initials are specified, a separate SI-index is created with the initials. (This permits the customer to later be accessed by its name or its initials). The customer number assigned is determined by reading the last record in the dataset using DBFIND mode 200 followed by DBGET mode 6 and incrementing it by one.

```
WORKING-STORAGE SECTION.
01 CUST.
    05 CUST-NUMBER      PIC 9(6).
    05 CUST-NAME        PIC X(20).
*   ...

01 CUST-INDEX.
    05 CUST-INITIALS    PIC X(6).
    05 CUST-NUMBER-X    PIC 9(6).

01 BASE             PIC X(8)   VALUE "  OEDB  ".
01 DSET             PIC X(16)  VALUE "CUST            ".
01 ITEM             PIC X(16)  VALUE "CUST-NUMBER     ".
01 LIST             PIC XX     VALUE "@".
01 INDEX-LIST       PIC XX     VALUE "!;".
01 ARGUMENT         PIC X(20).

01 STAT.
    05 CONDITION-WORD   PIC S9(4) COMP.
    05 STAT2            PIC S9(4) COMP.
    05 STAT3-4          PIC S9(9) COMP.
    05 STAT5-6          PIC S9(9) COMP.
    05 STAT7-8          PIC S9(9) COMP.
    05 STAT9-10         PIC S9(9) COMP.

01 MODE1            PIC S9(4) COMP VALUE 1.
01 MODE6            PIC S9(4) COMP VALUE 6.
01 MODE200         PIC S9(4) COMP VALUE 200.
01 DUMMY           PIC (X).
```

```
PROCEDURE DIVISION.

MAIN.
    PERFORM OPEN-BASE.

    CALL "DBFIND" USING BASE, DSET, MODE200, STAT, ITEM, DUMMY.
    IF CONDITION-WORD <> 0
        GO TO DB-ERROR.
    CALL "DBGET" USING BASE, DSET, MODE6, STAT, INDEX-LIST, CUST-NUMBER.
    ADD 1 TO CUST-NUMBER.
    DISPLAY "Enter customer name    :".
    ACCEPT CUST-NAME.
    DISPLAY "Enter customer initials:".
    ACCEPT CUST-INITIALS.

    CALL "DBPUT" USING BASE, DSET, MODE1, STAT, LIST, CUST.
    IF CONDITION-WORD <> 0
        GO TO DB-ERROR.

    IF CUST-INITIALS <> SPACE
        MOVE CUST-NUMBER TO CUST-NUMBER-X.

    CALL "DBPUTIX" USING BASE, DSET, MODE1, STAT, ITEM,
                        CUST-INDEX.
    IF CONDITION-WORD <> 0
        GO TO DB-ERROR.
```

Manually deleting SI-indices with DBDELIX

This example deletes the customer entry and all associated SI-indices that were added in the last example, using DBDELETE and DBDELIX.

```
WORKING-STORAGE SECTION.
01 CUST.
    05 CUST-NUMBER      PIC 9(6).
    05 CUST-NAME        PIC X(20).
*   ...

01 CUST-INDEX.
    05 CUST-INITIALS    PIC X(6).
    05 CUST-NUMBER-X    PIC 9(6).

01 BASE             PIC X(8)   VALUE "  OEDB  ".
01 DSET             PIC X(16)  VALUE "CUST            ".
01 ITEM             PIC X(16)  VALUE "CUST-NUMBER     ".
01 LIST             PIC XX     VALUE "@".
01 INDEX-LIST       PIC XX     VALUE "!;".
01 ARGUMENT         PIC X(20).

01 STAT.
    05 CONDITION-WORD  PIC S9(4) COMP.
    05 STAT2           PIC S9(4) COMP.
    05 STAT3-4         PIC S9(9) COMP.
    05 STAT5-6         PIC S9(9) COMP.
    05 STAT7-8         PIC S9(9) COMP.
    05 STAT9-10        PIC S9(9) COMP.

01 MODE1            PIC S9(4) COMP VALUE 1.
01 MODE5            PIC S9(4) COMP VALUE 5.
01 MODE7            PIC S9(4) COMP VALUE 7.
01 MODE10           PIC S9(4) COMP VALUE 10.
01 DUMMY            PIC (X).
```

```
PROCEDURE DIVISION.

MAIN.
    PERFORM OPEN-BASE.

    DISPLAY "Enter customer number to delete:".
    ACCEPT CUST-NUMBER.
    CALL "DBGET" USING BASE, DSET, MODE7, STAT, LIST, CUST-NUMBER.
    IF CONDITION-WORD <> 0
       IF CONDITION-WORD = 17
          DISPLAY "No entry" ·
          STOP RUN
       ELSE
          GO TO DB-ERROR.

    CALL "DBDELETE" USING BASE, DSET, MODE1, STAT.
    IF CONDITION-WORD <> 0
       GO TO DB-ERROR.

    MOVE ALL "?" TO CUST-INITIALS.
    MOVE CUST-NUMBER TO CUST-NUMBER-X.

    CALL "DBFIND" USING BASE, DSET, MODE10, STAT, ITEM, CUST-INDEX.
    IF CONDITION-WORD <> 0
       GO TO DB-ERROR.

    CALL "DBGET" USING BASE, DSET, MODE5, STAT, INDEX-LIST, CUST-INDEX, DUMMY.
    IF CONDITION-WORD <> 0 AND <> 15
       GO TO DB-ERROR.

    CALL "DBDELIX" USING BASE, DSET, MODE1, STAT, ITEM,
                        CUST-INDEX.
    IF CONDITION-WORD <> 0
       GO TO DB-ERROR
    ELSE
       STOP RUN.
```

Adding SI-dataset to the lock descriptor

This example shows the root SI-dataset being added to the *lock descriptor*, which already includes the CUST and SHIP datasets. The lock descriptor is used by DBLOCK *modes* 5 and 6.

```
WORKING-STORAGE SECTION.

01  DB-LOCK-DESCRIPTOR.
    05 LOCK-COUNT          PIC S9(4) COMP VALUE 3.
    05 LOCK-ENTRY.
        10 LOCK-1-LENGTH   PIC S9(4) COMP VALUE 22.
        10 LOCK-1-DSET     PIC X(16) VALUE "CUST;              ".
        10 LOCK-1-ITEM     PIC X(16) VALUE "@;                 ".
        10 LOCK-1-RELOP    PIC X(02)     VALUE "=".
        10 LOCK-1-VALUE    PIC X(08).
        10 LOCK-2-LENGTH   PIC S9(4) COMP VALUE 22.
        10 LOCK-2-DSET     PIC X(16) VALUE "SHIP;              ".
        10 LOCK-2-ITEM     PIC X(16) VALUE "@;                 ".
        10 LOCK-2-RELOP    PIC X(02) VALUE "=".
        10 LOCK-2-VALUE    PIC X(08).
        10 LOCK-3-LENGTH   PIC S9(4) COMP VALUE 22.
        10 LOCK-3-DSET     PIC X(16) VALUE "SI;                ".
        10 LOCK-3-ITEM     PIC X(16) VALUE "@;                 ".
        10 LOCK-3-RELOP    PIC X(02) VALUE "=".
        10 LOCK-3-VALUE    PIC X(08).
```

Calling SITRANSLATE

This example illustrates calling SITRANSLATE after accepting an argument. The code is the same as the partial-key retrieval example, except for the call to SITRANSLATE and the WORKING-STORAGE modifications.

```
WORKING-STORAGE SECTION.
01 CUST.
     05 CUST-NUMBER       PIC 9(6).
     05 CUST-NAME         PIC X(20).
     05 CUST-ADDRESS1     PIC X(20).
     05 CUST-ADDRESS2     PIC X(20).

01 SEARCH-NAME           PIC X(20).

01 BASE                  PIC X(8)  VALUE "  OEDB  ".
01 DSET                  PIC X(16) VALUE "CUST            ".
01 ITEM                  PIC X(16) VALUE "CUST-NAME       ".
01 LIST                  PIC XX    VALUE "@".
01 SITRANSLATE-AREA.
     05 OPERATOR            PIC X             VALUE SPACES.
     05 INPUT-ARGUMENT      PIC X(20)         VALUE SPACES.
     05 OUTPUT-ARGUMENT     PIC X(20)         VALUE SPACES.
     05 SITRANS-ERROR       PIC S9(4) COMP VALUE ZEROES.

01 STAT.
     05 CONDITION-WORD  PIC S9(4) COMP.
     05 STAT2           PIC S9(4) COMP.
     05 STAT3-4         PIC S9(9) COMP.
     05 STAT5-6         PIC S9(9) COMP.
     05 STAT7-8         PIC S9(9) COMP.
     05 STAT9-10        PIC S9(9) COMP.

01 MODE1                 PIC S9(4) COMP VALUE 1.
01 MODE5                 PIC S9(4) COMP VALUE 5.
01 DUMMY                 PIC X.
```

```
PROCEDURE DIVISION.

MAIN.
    PERFORM OPEN-BASE.

    DISPLAY "Enter customer name:".
*     Accept the input argument
    ACCEPT INPUT-ARGUMENT.

*   Make sure the OPERATOR and OUTPUT-ARGUMENT are initialized
    MOVE SPACES TO OPERATOR, OUTPUT-ARGUMENT.
*     The first three parameters must be passed using the "@" in COBOL
    CALL "SITRANSLATE" USING  @OPERATOR, @INPUT-ARGUMENT,
                              @OUTPUT-ARGUMENT, SITRANS-ERROR.
*   The SITRANS-ERROR should always be zero.  If not, display the
*   the error number and exit.

*   Notice in the call to DBFIND, use OUTPUT-ARGUMENT with no
*   modifications.
    CALL "DBFIND" USING BASE, DSET, MODE1, STAT, ITEM,
                        OUTPUT-ARGUMENT.
    IF CONDITION-WORD <> 0
       IF CONDITION-WORD =17
          DISPLAY "no entry"
          STOP RUN
       ELSE
          GO TO DB-ERROR.

    DISPLAY "Cust#      Name              Address".
    PERFORM GET-NEXT STAT5-6 TIMES.
    STOP RUN.

GET-NEXT SECTION.
    CALL "DBGET" USING BASE, DSET, MODE5, STAT, LIST, CUST, DUMMY.
    IF CONDITION-WORD <> 0
       GO TO DB-ERROR.
    DISPLAY CUST-NUMBER "  " CUST-NAME "  " CUST-ADDRESS1
       "  " CUST-ADDRESS2.

GET-NEXT-EXIT.
    EXIT.
```

SI-dataset structure

Between one and eight SI-datasets may be allocated for any database. Each dataset is a standalone detail set with the name **SI**, conditionally followed by the relative set number **1-7**.

The *root* SI-dataset contains the SUPERDEX definitions and must always exist--even if it does not contain any B-trees. It is normally named **SI**, but may alternately be named **SI0** if and only if a regular dataset named SI already exists.

If built by SIMAINT, the block size of each SI-dataset is equal to the database BLOCKMAX at the time the set is created.

The SI-datasets must appear consecutively in the dataset list in continuous numeric order starting with **SI** (i.e. **SI** or **SI0** immediately followed by **SI1**, **SI2**, etc.).

Additionally, write access must be granted to all user classes configured for write access to any SUPERDEX'ed dataset in the database.

SI-dataset capacity

The capacity of the SI-dataset is based on the space required to store the SI-indices, which depends on the actual value of the SI-keys. The calculation of the recommended SI-dataset capacity is performed by the SIMAINT utility under the assumption that the values of all SI-keys are different--the worst case condition. If there are many occurrences of the same SI-key values--especially for keyworded SI-paths--the actual space requirements may be considerably less than that calculated.

If you foresee this situation and there are B-trees for several SI-paths stored in a single SI-dataset, you may override the recommended capacity and specify one that is lower. It is safer, though, to use the recommended capacity and reduce it after the SI-indices have been generated. In fact, SIMAINT verifies that the SI-dataset capacities are sufficiently high based on the worst-case calculation.

☞ It is also important to note that the SI-dataset capacities calculated by SIMAINT do not allow for future capacity changes in the datasets they index nor additional SI-paths that may be added at some later time. You may want to specify higher SI-dataset capacities to leave room for dataset capacity increases and additional SI-paths.

SIMAINT initially generates B-trees that are optimized for space utilization. When performing heavy updates to the SI-indices, this optimization may be lost and the B-tree must expand. Therefore, it is recommended that 20 percent free space be left in each SI-dataset to accommodate this situation. The extra space utilized can always be regained by reorganizing all the SI-paths in the SI-dataset.

SI-item

The item named **SI** which is the only field in each SI-dataset.

The SI-item is built as a compound item by the SIMAINT.PUB utility program in the format $nX254$, where n is the subitem count as determined by the block size of the SI-dataset, with a maximum value of 16.

Alternately, as some software systems do not permit compound items, multiple individual items may be defined instead. These items should be named SI1 - SIn, with n the same as the subitem count.

Additionally, write access must be granted to all user classes configured for write access to any SUPERDEX'ed dataset in the database.

SI-index base

A separate database which may be optionally used to maintain the SI-indices, rather than locating them in the primary database with the data entries. If utilized, SI-datasets starting with **SI1** reside in the SI-index base; the root SI-dataset (**SI** or **SI0**), which contains only the SI-definitions, must reside in the primary database. Both the primary and SI-index base must contain the SI-item.

The SI-index base must reside in the same group and account as the primary base. Its name must be the same as the primary base with **SI** as the last two characters. For database names of 4 characters or less, append **SI**; for 5 or 6 character database names, replace the last one or two characters with **SI**. For example, the SI-index base for the base OEDB would be **OEDBSI**; for CUSTDB, it would be **CUSTSI**.

The SI-index base may contain additional items and datasets in addition to the SI-item and SI-dataset(s), and the SI-datasets must be in consecutive order. The block size and blocking factor of the SI-dataset(s) must be the same as that of the root SI-dataset in the primary base. Additionally, the SI-index base must have the same user classes and passwords as the primary base.

SI-index

The SI-index is comprised of an SI-key followed by an SI-extension. The SI-index differs for SI-paths related to master and detail datasets.

For master sets, the SI-index consists of up to three SI-subkeys plus the IMAGE search field value, as shown:

SI-subkey 1	SI-subkey 2	SI-subkey 3	IMAGE search field value
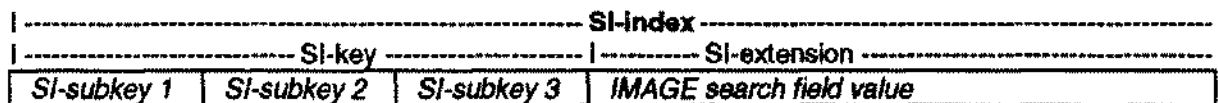
The search field may be specified as the last significant SI-subkey of a concatenated SI-key. In this case, the search field value is contained only once and is used for both selection and indexing.

For detail sets, up to four SI-subkeys are allowed, followed by a double-word relative record number:

```
|---------------------------------------------SI-index----------------------------------------------|
|-------------------------- SI-key --------------------------------| ------- SI-extension ------------------|
```

SI-subkey 1	SI-subkey 2	SI-subkey 3	SI-subkey 4	Detail relative record number

There are compression techniques that are used to save space when there are repeating SI-key values. For example, the SI-key value will not be stored twice if there are two records with the same SI-key and different master key values or relative record numbers.

Additionally, no duplicate SI-indices are stored. This means that if a detail record has a keyworded key, if a word is repeated in the field, there will only be one SI-index for that record.

SI-pointer

The SI-pointer consists of the last SI-index accessed plus one bit indicating whether the pointer is located in front of or after the current SI-index.

SI-subset

An extra data segment (XDS) that contains the results of a DBFIND performed in relational access mode. Ordinarily, the SI-subset contains only the SI-extensions that map the qualifying entries, as shown:

Master set:

IMAGE search field value

Detail set:

relative record number
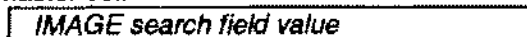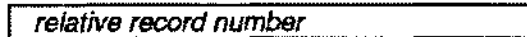
The contents of the SI-subset are used for comparison with subsequent DBFIND calls.

If an SI-link is specified in addition to the SI-path in the DBFIND *item* parameter, the SI-subset also contains the value of the SI-link, as shown:

Master set:

SI-link	I	IMAGE search field value

Detail set:

SI-link	I	relative record number

In this case, both the SI-link and SI-extension are used for comparison. Also, the value of the SI-link is used to determine the sorting order when entries are returned.

The following table identifies SUPERDEX's internal limits. Most limits are not checked, and results when exceeded are unpredictable.

SUPERDEX maximum limits

Facility	Maximum Limit
Number of Datasets per database with SI-paths	198
Number of SI-paths per database (with SI-index length under 30 words)	400
Number of SI-paths per database (with SI-index length of 99 words)	270
Number of SI-paths per dataset (with an SI-item length of 508 words)	22
Number of SI-paths per dataset (with an SI-item length of 1016 words)	53
SI-index length	127 words
SI-subkey length	63 words
Number of SI-indices per SI-dataset (with average SI-index length of 10 words)	102,400,000
Number of SI-indices per SI-dataset (with average SI-index length of 30 words)	68,266,000
Number of SI-indices per SI-dataset (with average SI-index length of 99 words)	20,686,000
Number of keywords per simple SI-key (or first SI-subkey when concatenated)	16
Number of items allowed in a grouped SI-path	32
Number of words in the keyword exclude path (with length of 4 words)	18,000
Total number of qualified entries in both SI-subsets (without SI-link): Detail dataset	300,000
Total number of qualified entries in both SI-subsets (without SI-link): 5 word Master key (i.e. X10) 2 word Master key (i.e. I2)	200,000 300,000

Error and exceptional conditions

SUPERDEX intrinsic error and exceptional conditions

SUPERDEX returns standard IMAGE condition codes and messages upon encountering an error or exceptional condition. These condition codes and messages describe SUPERDEX conditions that are equivalent to IMAGE conditions.

Also, because SUPERDEX uses standard IMAGE intrinsics to manage its B-tree structures, an error may indicate a problem in the SI-dataset rather than the dataset referenced by the *dset* parameter. Additionally, if SUPERDEX has been configured to maintain its SI-indices in a separate SI-index base, an error may indicate a problem in the SI-index base rather than the base referenced by the *base* parameter.

Some of the more common and noteworthy condition word values that may be returned by various intrinsics in accessing a SUPERDEX'ed base and what they mean are shown in the **SUPERDEX intrinsic error and exceptional conditions table** on the following pages.

SUPERDEX utility error and exceptional conditions

The **SUPERDEX utility error and exceptional conditions** table lists the various error messages that could be issued by the SIMAINT utility program, their meanings, and their corrective actions.

Program failures related to SUPERDEX

Programs that are run through the SUPERDEX SL or XL require certain capabilities and sufficient stack; otherwise, an error will occur. The **Program failures related to SUPERDEX** table lists these errors, their causes, and remedies.

SUPERDEX intrinsic error and exceptional conditions

Type	Condition word/description
Message	**-21 BAD PASSWORD**
Intrinsic	DBOPEN
Meaning	Inconsistency in SI-definitions.
Action	Use SIMAINT,STRUCT against the database.
Message	**-31 BAD (UNRECOGNIZED) DBFIND MODE: xxx**
Intrinsic	DBFIND mode 1nn or 2nn
Meaning	The length imposed by the specified *mode* exceeds the length of the *argument* value.
Action	Specify a mode that does not exceed the *argument* length.
Message	**-41 DBUPDATE WILL NOT ALTER A SEARCH OR SORT ITEM**
Intrinsic	DBUPDATE
Meaning	Database was opened in *mode* 2 and an update against one or more SI-key was attempted.
Action	Use a DBOPEN mode other than *mode* 2 if updating SI-keys.
Message	**-52 ITEM SPECIFIED IS NOT AN ACCESSIBLE SEARCH ITEM IN THE SPECIFIED SET**
Intrinsic	DBFIND
Meaning	The specified *item* is neither an IMAGE key or a SUPERDEX SI-path.
Action	Use SIPATH to show the configured IMAGE keys and SUPERDEX SI-paths.
Message	**-53 DBPUT LIST IS MISSING A SEARCH OR SORT ITEM**
Intrinsic	DBPUT
Meaning	All IMAGE keys and SUPERDEX SI-keys are not included in the *list* parameter.
Action	Change the *list* to include all IMAGE keys and SUPERDEX SI-keys.
Message	**10 BEGINNING OF FILE**
Intrinsic	DBGET *mode* 16
Meaning	After calling DBFIND *mode* 100 or 200 and DBGET *mode* 16, the entry with the lowest alphabetic SI-key in the dataset has been returned.
Action	Depends on the program design.
Message	**11 END OF FILE**
Intrinsic	DBGET *mode* 15
Meaning	After calling DBFIND *mode* 100 or 200 and DBGET *mode* 15, the entry with the highest alphabetic SI-key in the dataset has been returned.
Action	Depends on the program design.
Message	**14 BEGINNING OF CHAIN**
Intrinsic	DBGET *mode* 5 or 15
Meaning	In *mode* 5, the entry with the lowest alphabetic SI-key that matches the specified DBFIND *argument* has been returned. In *mode* 15, the entry with the lowest alphabetic SI-key in the dataset has been returned.
Action	Depends on the program design.

Message	**15 END OF CHAIN**
Intrinsic	DBGET *mode* 6 or 16
Meaning	In *mode* 6, the entry with the highest alphabetic SI-key that matches the specified DBFIND *argument* has been returned. In *mode* 16, the entry with the highest alphabetic SI-key in the dataset has been returned.
Action	Depends on the program design.
Message	**16 THE DATA SET IS FULL**
Intrinsic	DBPUT
Meaning	Either the dataset referenced in the *dset* parameter or the SI-dataset that corresponds with the *dset* is full (the *dset* with be displayed in either case).
Action	Increase the capacity of the *dset*, or the SI-dataset.
Message	**17 THERE IS NO CHAIN FOR THE SPECIFIED SEARCH ITEM VALUE**
Intrinsic	DBFIND
Meaning	1. No entry exists that matches the specified *argument*. 2. An SI-index that has no corresponding data record was detected.
Action	1. Even though an error is returned, if called in *mode* 1*nn* or 2*nn*, the internal SI-pointer is set and DBGET *mode* 15 and 16 may be used to retrieve the entries with SI-keys greater than and less than the specified *argument*. 2. Reorganize the suspected corrupt SI-path using SIMAINT.
Message	**18 BROKEN CHAIN - FORWARD AND BACKWARD POINTERS NOT CONSISTENT**
Intrinsic	DBGET
Meaning	Possible inconsistency in B-tree, due to program abort or system failure
Action	Reorganize the SI-paths related to the suspected dataset, or all the SI-paths in the database.
Message	**60 DATABASE ACCESS DISABLED**
Intrinsic	DBOPEN
Meaning	The copy of SUPERDEX has expired, or the database has not been stamped by a new version of SIMAINT.
Action	If a current version SUPERDEX is available, either demo or permanent, run SIMAINT against the database, otherwise delete the SI-item and SI-dataset(s) from the database.

SUPERDEX utility error and exceptional conditions

Type	Description
Message	**CAPACITY EXCEEDS MPE LIMIT - SI DATASET CANNOT BE CREATED**
Meaning	The capacity calculated for a newly defined SI-dataset requires a worst-case dataset file that exceeds the MPE limits.
Action	Specify additional SI-dataset as necessary, or specify a smaller capacity size.
Message	**CAPACITY OF SIn DATASET NOT SUFFICIENT - NO LOADING RECOMMENDED CAPACITY FOR SIn: xxxxxxxx**
Meaning	The capacity of the current SI-dataset is not high enough to accommodate the new SI-indices, so the SI-path configuration is saved but no SI-indices are generated. Change the capacity of the SI-dataset indicated to at least the recommended
Action	capacity, and run SIMAINT to populate the related SI-paths.
Message	**COMPOUND ITEM NOT ALLOWED HERE**
Meaning	A compound item was specified for an SI-subkey as other than the first SI-subkey
Action	Compound items may only be used as the first SI-subkey in an SI-key
Message	**CREATOR ACCESS REQUIRED**
Meaning	You are not logged on as the creator of the database you are attempting to access.
Action	Log on as the creator of the database and rerun the process.
Message	**DEFINITIONS CANCELLED FOR CURRENT DATASET**
Meaning	\ was entered after defining an SI-path for a dataset, so any SI-paths defined for this dataset in the current run of SIMAINT are flushed
Action	None (status message only)
Message	**EXTENSION FAILED - DATABASE IS OK**
Meaning	An error was detected during processing, but the database was not damaged.
Action	Correct the condition that caused the error and rerun SIMAINT.
Message	**EXTENSION FAILED - PLEASE GO TO BACKUP OF DATABASE**
Meaning	An error was detected during the extension phase of SIMAINT and the database is damaged.
Action	Correct the error condition displayed, restore the database from backup, and rerun SIMAINT.
Message	**FILE ERROR ACCESSING SI-DATASET**
Meaning	An MPE file system error was detected while accessing the newly-created SI-dataset.
Action	Correct the error condition displayed, and rerun SIMAINT.
Message	**FILE ERROR ACCESSING NEW ROOT FILE**
Meaning	An MPE file system error was detected while accessing the newly-rebuilt database root file.
Action	Correct the condition that caused the error, and rerun SIMAINT.
Message	**FILE ERROR ACCESSING OLD ROOT**
Meaning	An MPE file system error was detected while accessing the current database root file.
Action	Correct the condition that caused the error, and rerun SIMAINT.
Message	**GROUPING NOT ALLOWED FOR THIS SI-PATH**
Meaning	An independent SI-path was configured as grouped.
Action	This configuration is illegal.

Message	**NONEXISTENT DATASET**
Meaning	A dataset name was specified that does not exist in the database.
Action	Enter **?** for a list of datasets.
Message	**ILLEGAL OPTION**
Meaning	An unrecognized suffix was specified on either the database, dataset or the SI-path name.
Action	Valid database suffixes are **/1**, **/2**, and **/3**. Valid dataset suffixes are **/1** - **/7**, **/D**, and **/R**. Valid SI-path suffixes are **/B**, **/D**, **/G**, **/K**, and **/R**.
Message	**ILLEGAL OPTION - NO CORRESPONDING SI-DATASET EXISTS**
Meaning	Either the referenced SI-dataset (specified via *dataset/n*) does not exist, or the SI-datasets do not appear in numerical consecutive order in the database.
Action	The assigned SI-dataset does not exist and SIMAINT.PUB was not executed, so the SI-dataset must be built by hand. In the latter case, the SI-datasets must appear in sequential order.
Message	**INPUT ERROR READING DATASET**
Meaning	A file system error was detected when reading the dataset.
Action	Print a copy of the error tombstone and call Bradmark Technical Support ☎
Message	**INPUT SORTLIB: TOO MANY RECORDS**
Meaning	The *average number of keywords* specified in configuring a keyworded SI-path or *average number of indices* for a custom SI-path is not high enough.
Action	Use SIMAINT to reorganize the SI-path, and specify a higher average number of keywords or indices.
Message	**INVALID SI-DATASET**
Meaning	1. For a database with one or more SI-paths, the root SI-dataset has been corrupted. 2. For a database with no SI-paths, a regular dataset named **SI** already exists.
Action	1. Erase the root SI-dataset using a utility (i.e. **DBGENERAL**) or delete all its entries, and redefine all SI-paths for the database. 2. Configure SUPERDEX with a root SI-dataset of **SI0** (refer to *Section 3 Configuration/Establishing SI-indices* for details).
Message	**ITEM NOT TYPE U OR X - SI-PATH CANNOT BE KEYWORDED**
Meaning	The specified item may not be configured as the first SI-subkey in a keyworded SI-path because it is not alphanumeric (data type U or X).
Action	Numeric data types are not supported for keywording.
Message	**ITEM NOT IN DATASET**
Meaning	The specified item does not exist in the current dataset.
Action	Enter **?** for a list of items in the current dataset.
Message	**LENGTH CONFLICT IN GROUP**
Meaning	The item specified for grouping is of a longer length than other items in the group.
Action	Configure the longest SI-key in the group first.
Message	**MAXIMUM OF 199 SETS EXCEEDED**
Meaning	Creating the configured SI-dataset(s) would cause the database to have more than 199 dataset (99 for non-Turbo IMAGE databases).
Action	Specify fewer SI-datasets.

Message	**NO SI-PATHS DEFINED FOR SPECIFIED DATASET**
Meaning	A dataset was specified with a suffix of **/D** or **/R**, but no SI-paths are related to the dataset
Action	**/D** and **/R** are only allowed on datasets with existing SI-paths.
Message	**ODD NIBBLES NOT ALLOWED**
Meaning	The specified item is of data type P and its subitem length is odd.
Action	The subitem length for type P items must be even.
Message	**SI-PATH ALREADY EXISTS**
Meaning	An SI-path with the specified name already exists for the dataset.
Action	Specify a unique SI-path name or append **/G** to group the SI-path.
Message	**SI-PATH DOES NOT EXISTS**
Meaning	An SI-path that does not exist in this database was specified with either **/D, /G,** or **/R.**
Action	Specify the name of an existing SI-path.
Message	**SORTLIB ERROR: TOO MANY INPUT RECORDS**
Meaning	The *average number of keywords* specified in configuring a keyworded SI-path or *average number of indices* for a custom SI-path is not high enough.
Action	Reorganize the SI-path, and specify a higher average number of keywords or indices.
Message	**TOO MANY SI-PATHS DEFINED FOR** *dataset*
Meaning	More SI-paths have been specified for a related dataset than the maximum limit.
Action	Refer to the table in the *Appendix C Maximum Limits*, or call Bradmark Technical Support for information on how to define more. ☎
Message	**TURBO IMAGE IS REQUIRED TO EXTEND DATABASE**
Meaning	SIMAINT.PUB was run against an non-Turbo IMAGE database, and there were no SI-paths defined previously.
Action	Add the SI-item and SI-dataset(s) manually and then rerun SIMAINT to configure the SI-paths, or convert the non-Turbo IMAGE database to a TurboIMAGE structure.
Message	**TYPE CONFLICT IN GROUP**
Meaning	The item specified for grouping is of a different data type than the already existing SI-path.
Action	All items in a group must be of the same data type.
Message	**WARNING: ILR ENABLED**
Meaning	Non-critical message indicating the ILR is enabled for the database when SIMAINT is executed, and therefore processing may be slower.
Action	In the future, disable ILR prior to executing SIMAINT.
Message	**WARNING: LOGGING ENABLED**
Meaning	Non-critical message indicating that logging is enabled for the database when SIMAINT is executed, and therefore processing may be slower.
Action	In the future, disable logging prior to executing SIMAINT.
Message	**WARNING: RUN WITHOUT ;LIB=G OR ;LIB=P**
Meaning	Non-critical message indicating that SIMAINT was run with **;LIB=G** or **;LIB=P**, therefore processing may be slower.
Action	Run SIMAINT without specifying any external libraries.

Message	WARNING! INCONSISTENCY DETECTED. RERUN WITH ,STRUCT
Meaning	An inconsistency between the database structure and the SUPERDEX configuration was detected.
Action	Run SIMAINT,STRUCT against the database.
Message	WARNING! OPTION IGNORED, PREVIOUS DEFINITION RETAINED
Meaning	The specified SI-dataset option does not match the existing SI-dataset option for the dataset.
Action	Once an SI-path has been defined for a dataset, the SI-dataset option can not be modified. To change the SI-dataset, the existing SI-path(s) must be deleted and reconfigured with the new SI-dataset option specified.

Program failures related to SUPERDEX

Type	Error/Description
Message	**STACK OVERFLOW (PROGRAM ERROR #20)**
Cause	Available program stack space is insufficient for SUPERDEX access.
Remedy	Increase stack by 2 - 3 Kwords or more.
Message	**PROCESS QUIT; PARM = 61 (PROGRAM ERROR #18)**
Cause	Available program stack space is insufficient for SUPERDEX access.
Remedy	Increase stack by 2 - 3 Kwords or more.
Message	**PROCESS QUIT; PARM = 62 (PROGRAM ERROR #18)**
Cause	Exceeds available SUPERDEX table
Remedy	Call Bradmark Technical Support. ☎
Message	**PROCESS QUIT; PARM = 63 (PROGRAM ERROR #18)**
Cause	Exceeds available SUPERDEX table
Remedy	Call Bradmark Technical Support. ☎
Message	**PROCESS QUIT; PARM = 64 (PROGRAM ERROR #18)**
Cause	Exceeds available SUPERDEX table
Remedy	Call Bradmark Technical Support. ☎
Message	**ILLEGAL CAPABILITY (LOADER ERROR #39)**
Cause	Group and/or account where program resides lacks capabilities granted to program.
Remedy	Determine which capabilities group and/or account lack and add them.
Message	**ILLEGAL CAPABILITY (RUN-TIME ERROR #2)**
Cause	Program referencing SUPERDEX SL or XL lacks DS and conditionally MR capability.
Remedy	Add DS capability and, if required by locking strategy, MR capability to program.

Business Basic interface

Installation

BUSINESS BASIC does not call IMAGE intrinsics directly; rather, it calls its own special intrinsics in the system SL or XL (e.g. BB_DBPUT) which in turn call the IMAGE intrinsics. For this reason, it is not sufficient to simply run programs to access the SUPERDEX SL/XL, since their IMAGE externals will still be resolved in the system SL/XL and the SUPERDEX intrinsics will be bypassed.

The solution is to place both the BUSINESS BASIC intrinsics and SUPERDEX intrinsics in the same group or account SL or XL. This way, the BUSINESS BASIC intrinsics call the SUPERDEX intrinsics rather than the standard IMAGE intrinsics.

Unfortunately, this is often a somewhat involved process, since HP does not provide a USL containing the BUSINESS BASIC intrinsics. It is therefore necessary to make a copy of the system SL/XL, purge all the segments that are not reserved for BUSINESS BASIC, add in the SUPERDEX intrinsics segments, and then use this new SL/XL as the SUPERDEX SL/XL.

The following job stream creates an SL that contains both the BUSINESS BASIC and SUPERDEX intrinsics in a new group BBASIC.SUPERDEX, which it builds. Before streaming it, :RELEASE SL.PUB.SYS and then :SECURE it when done.

```
!JOB MAKEBBSL,MGR.SUPERDEX;OUTCLASS=,1
!CONTINUE
!PURGEGROUP BBASIC
!NEWGROUP BBASIC
!SEGMENTER
SL SL.PUB.SYS
COPYSL 1,SL.BBASIC;USERFORMAT
EXIT
!BUILD LIST.BBASIC;REC=-80,1,F,ASCII;DISC=20000,32,8;DEV=DISC
!FILE SEGLIST=LIST.BBASIC,OLD;SAVE
!SEGMENTER
SL SL.BBASIC
LISTSL
EXIT
!PCOPY FROM=LIST.BBASIC;TO=PURGESL.BBASIC;NEW;SUBSET="SEGM",1
!PURGE LIST.BBASIC
```

```
!EDITOR
TEXT PURGESL.BBASIC
CHANGEQ 29/80," ",ALL
CHANGEQ 1,"PURGESL ",ALL
CHANGEQ 16/20,",",ALL
ADDQ .1
SL SL.BBASIC
//
ADDQ
CLEANSL
USL SIUSL.PUB.SUPERDEX
ADDSL SIPROC0
ADDSL SIPROC1
ADDSL SIPROC2
ADDSL SIPROC3
ADDSL SIPROC4
COPYSL 1
EXIT
//
LISTQ 1
WHILE
FINDQ "HPBB"
DELETEQ *
KEEP PURGESL.BBASIC,UNN;EXIT
!RUN SEGDVR.PUB.SYS;STDIN=PURGESL.BBASIC
!PURGE PURGESL.BBASIC
!EOJ
```

Once created, copy the new SL into the account in which the object programs reside, either in the PUB group or the group in which the programs reside.

☎ For Native Mode Business Basic, please contact Bradmark's Technical Support for information on copying the XL.

Running programs

All programs that access databases that have been modified for SUPERDEX access must be run through an SL or XL that contains both the BUSINESS BASIC intrinsics and the SUPERDEX intrinsics, using **;LIB=P** (if the SL/XL is in the PUB group) or **;LIB=G** (if the SL/XL is in the object program group).

For programs run interactively, HPBB.PUB.SYS and HPBBCNFG.PUB.SYS must be copied to a group/account in which the SUPERDEX SL or XL resides and HPBB run with **;LIB=P** or **;LIB=G**.

Adding, updating, and deleting entries

BUSINESS BASIC programs are run through SUPERDEX's SL or XL, so that entries added, updated, and deleted from BUSINESS BASIC programs also cause the SI-indices to be added, updated, and deleted automatically.

Qualifying and retrieving entries

Entries are qualified and retrieved in BUSINESS BASIC using the same methods as for other 3rd generation languages, with SUPERDEX's DBFIND and DBGET.

Because BUSINESS BASIC's DBFIND does not have a *mode* parameter, it always calls SUPERDEX's DBFIND in *mode* 1. Fortunately, the majority of SUPERDEX retrievals can be accomplished with only DBFIND *mode* 1.

For retrievals that can only be performed by SUPERDEX using a *mode* other than 1, such as approximate match retrieval, SUPERDEX's DBFIND intrinsic may be called as an external procedure.

☞ **SUPERDEX does not support the floating-point decimal elements as SUPERDEX Path Items for Native Mode Business Basic at this time.**

TRANSACT interface

Installation

TRANSACT.PUB.SYS is copied to a group/account where the SUPERDEX SL or XL resides and run with `;LIB=G` or `;LIB=P`. UDCs, menus, job streams, etc... must be changed accordingly.

If using Native Mode TRANSACT/XL, it is also necessary to duplicate the module named **HP30138** from XL.PUB.SYS into the SUPERDEX XL. This can be facilitated using the COPYXL command of the LinkEditor.

It is recommended that SI-paths be defined in the data dictionary, although they may alternately be defined in programs that access them.

If defining SI-paths in the dictionary, they should not be specified as elements in any files. Each SI-path should be configured as **TYPE X** (alphanumeric) with a **SIZE** long enough to accommodate the SI-key length plus operators and multiple values. It is recommended that SI-paths be identified using an appropriate **DESC**ription to distinguish them from items.

Adding, updating, and deleting entries

TRANSACT is run through SUPERDEX's SL or XL, so that entries added, updated, and deleted from TRANSACT programs also cause the SI-indices to be added, updated, and deleted automatically.

Using (CHAIN)

TRANSACT has the capability of updating or deleting entire detail data sets using the **(CHAIN)** option. Because SUPERDEX modifies the IMAGE pointers, it is possible to confuse IMAGE during a **(CHAIN)** process, especially when doing PROCEDUREs within the **(CHAIN)** process. Therefore, it is recommended NOT to use the **(CHAIN)** option for **UPDATE**s or **DELETE**s. A **FIND (CHAIN)** should be used to find the chain, followed by individual record UPDATEs.

Qualifying and retrieving entries

Entries are qualified and retrieved using the same method as chained retrieval of entries in detail datasets. The SI-path name is placed into the *key* register and the SUPERDEX search argument into the *argument* register.

If SI-paths were not defined in the dictionary, define each SI-path in the program, for example:

```
DEFINE(ITEM):
   CUSTOMER-NAME-KW   X(40):
   ADDRESS-CITY-KW    X(40):
   PART-ORDER         X(20);
```

Then, place the search argument (as defined for SUPERDEX's DBFIND intrinsic) into the *argument* register by moving it to the item that represents the SI-path, as shown:

```
MOVE SI-path = "argument";
```

Then, place the SI-path name to be accessed into the *key* register, as shown:

```
SET(KEY) LIST(SI-path);
```

To retrieve entries in ascending sorted order:

```
FIND(CHAIN) dataset;
```

or in descending order:

```
FIND(RCHAIN) dataset;
```

To return the number of qualifying entries in the *status* register, use the same technique as for an IMAGE path:

```
PATH dataset;
```

This same method is used when performing relational access against multiple SI-paths and datasets: just set the *key* and *argument* registers as shown and perform one or more **PATH** commands as necessary against the dataset(s). For example, when using the RPN boolean format, make sure to enclose values in square brackets and append the appropriate boolean operator as required when performing relational access.

Match register

Entries that are retrieved using an SI-path may be restricted using the *match* register, as for IMAGE paths. It may, however, be more efficient to instead configure the SI-path to use a concatenated SI-key that includes the field being matched on as an SI-subkey. In this case, the concatenated value (for both fields) would be specified in the *argument* register. The advantage is that only the entries that match on both fields would be initially selected, rather than selecting entries that qualify on one field and filtering out those that do not match on the other.

LIST change

TRANSACT programs (including those compiled with FASTRAN) use the ＊ *list* once the *list* is first established. SUPERDEX may change the *list*, especially when updating an entry. For this reason, it is recommended to include all the fields that are used as SI-keys in the *list* or use the @ *list* when updating entries.

PROC DBFIND

The vast majority of SUPERDEX retrievals can be performed using the methods described, which access SUPERDEX's DBFIND intrinsic in *mode 1*.

For retrievals that can only be performed by SUPERDEX using a *mode* other than 1, such as approximate match retrieval, SUPERDEX's DBFIND intrinsic may be called as an external procedure using **PROC DBFIND**. Once this has been called, qualifying entries can be retrieved using:

```
FIND(CHAIN) dataset, LIST=(@), STATUS;
```

FASTRAN

The FASTRAN compiler from Performance Software Group may be used to compile and prepare
TRANSACT programs that use SUPERDEX. These programs would be run through the SUPERDEX
SL or XL using ;LIB=G or ;LIB=P, just as interpreted TRANSACT programs. Additionally,
existing object programs would require that DS and MR capability be added, which could be done
with the **ALTCAP.PUB.FASTRAN** program.

Alternately, the SUPERDEX procedures may be prepared directly into FASTRAN object programs,
which would be run without specifying ;LIB=G or ;LIB=P. In this case, DS and MR capability
would be included in the program capability list on the :PREP. To do so, with the FASTRAN UDCs
(**UDC.PUB.FASTRAN**) set locally:

```
:FASTCOMP source,,$NULL,options              << creates USL in $OLDPASS >>

:FASTSEG                                      << FASTRAN segmenter >>
=MAIN systemname,$OLDPASS
=INCLUDE FSTN''LIBC',LUSL.PUB.FASTRAN
=INCLUDE SIPROC0,SIUSL.PUB.SUPERDEX
=INCLUDE SIPROC1,SIUSL.PUB.SUPERDEX
=INCLUDE SIPROC2,SIUSL.PUB.SUPERDEX
=INCLUDE SIPROC3,SIUSL.PUB.SUPERDEX
=INCLUDE SIPROC4,SIUSL.PUB.SUPERDEX
=PREP object;CAP=IA,BA,DS,MR
=EXIT
:SAVE object
```

NetBase interface

Installation and configuration

SUPERDEX is installed and configured as documented in the **SUPERDEX User Manual**, with the following restrictions.

SUPERDEX is installed for each database on each system using a separate SI-index database to maintain the SI-indices rather than maintaining the SI-indices in the primary database. Refer to the *Separate database for SI-indices* chapter in the *Configuration/Establishing SI-indices* section of the **SUPERDEX User Manual** for instructions.

The SUPERDEX SL is located at the group level in the same group as object programs, with the NetBase SL located at the account level (PUB group). Programs are run with **;LIB=G**.

If programs are already located in the PUB group, it is necessary to move them to a different group in which the SUPERDEX SL resides. NetBase's SCANJOB program or MPEX's %EDIT command may be used for changing **"LIB=P"** to **"LIB=G"** in job streams and UDCs.

Note that due to the way in which NetBase handles the CREATEPROCESS intrinsic, any program which is created as the son process of another program is automatically directed to the same SL as the father program--regardless of the LIBSEARCH parameter used in the CREATEPROCESS call.

SIGROUPSL JCW

In order for SUPERDEX to access the NetBase procedures in the account SL, it is necessary to set a special JCW, as shown:

```
:SETJCW SIGROUPSL = 1
```

Unless the **SIGROUPSL** JCW is set, SUPERDEX will bypass the NetBase procedures altogether.

It is recommended that the SETJCW command be included in a system wide logon UDC to make sure that it is set for all users at all times.

Shadowing

In enabling databases for shadowing, only each primary database which contains the data is enabled. Shadowing is not enabled on any SI-index base.

A special version of the **NBPOST** program which has been **:PREP**ped with the SUPERDEX procedures is run on the remote system to automatically perform the necessary SUPERDEX indexing on the remote system whenever entries on the local system are added, updated, and deleted.

Network File Access

SUPERDEX procedures are resolved on the local system and the resulting IMAGE procedure calls are transported to the remote system where they are resolved by IMAGE.

INDEX

B (Continued)

C

D

M (Continued)

N

O

P

Q

R

S (Continued)

S (Continued)