HowMessy 2.9

User Manual

by Robelle Solutions Technology Inc.

Robelle Solutions Technology Inc.

Suite 372, 7360 137 Street
Surrey, BC Canada V3W 1A3

Phone: 604.501.2001

Fax: 604.501.2003

E-mail: sales@robelle.com

E-mail: support@robelle.com

Web: www.robelle.com

Contents

Chapter 1 - Welcome to HowMessy

Introduction

Welcome to version 2.9 of HowMessy -- the fast report on the internal efficiency of your database.

HowMessy answers the question, "How messy are things inside my database?" HowMessy shows percentage secondaries, worst cluster, longest chains, average chains, blocks per chain, percentage inefficient pointers, and deviation from maximum efficiency. With this information, you can decide what database changes might improve performance: increase or decrease capacity, add or delete paths, remove sort fields, alter primary paths, or repack a dataset along the primary path.

HowMessy works with all versions of IMAGE, including TurboIMAGE and IMAGE/SQL. HowMessy checks the database type and automatically adjusts to the type of database (e.g., TurboIMAGE, IMAGE/SQL, Jumbo datasets).

Authorization to Use HowMessy

HowMessy is a Bonus program that accompanies most Robelle products. We will send you a copy if your Robelle order entitles you to receive it. As a Robelle customer you may install HowMessy on your CPUs which are licensed for our products. There is no charge for using HowMessy, but you are <u>not</u> free to distribute it. Please install Howmessy only on your licensed CPUs.

HowMessy is in the PUB group to remind you that it is a Bonus product, and not part of any library that you can freely distribute. There is another category of programs from Robelle, called the "Qlib". These are programs which you may use on any CPU and may distribute freely to your friends. HowMessy is <u>not</u> part of the Qlib.

If you have questions on whether you are authorized to install HowMessy on a particular CPU, please call us for advice.

Compatibility-Mode and Native-Mode

HowMessy is available in compatibility-mode for MPE IV and MPE V and in native-mode for MPE/iX. The native-mode version of HowMessy is much faster than the compatibility-mode version on MPE/iX. Both versions of HowMessy have exactly the same features and produce the same report.

MPE/iX

Users often wonder if the performance problems highlighted by HowMessy apply to IMAGE/SQL databases on MPE/iX. While MPE/iX is very different from MPE V, the ideas behind the HowMessy report still apply to MPE/iX IMAGE/SQL databases. HowMessy shows how much extra disc I/O may occur in a messy database. This potential disc I/O affects performance on both MPE V and MPE/iX.

Documentation

HowMessy User Manual and Change Notice are available on line at:

http://www.robelle.com/library/manuals/

User Manual

The user manual contains the full description of all the HowMessy commands, as well as usage tips. The manual is up-to-date with all the latest changes incorporated in HowMessy. To see only the changes in the latest version, see the "What's New" section of the manual, or see the change notice.

Highlights in 2.9

- HowMessy now supports databases with that utilize the new limits of 1200 items, 240 datasets and 64 pasts from a master.
- HowMessy will report an error if an attempt to analyze a database that has been converted or built with the new entrynumber format.
- HowMessy would fail with a Speed Demon prefetch error on small datasets that had a block size that was not on a 128 word boundary.
- HowMessy would fail on certain older versions of Image and the CM version would fail on Classic machines.

Dynamic Dataset Expansion (MDX and DDX)

You can enable dynamic expansion of datasets for an Image/SQL database using a Third Party Tool like Adager on an existing database or by using DBSCHEMA settings for a new database. If you have enabled either MDX or DDX, HowMessy reports ADX, MDX, or DDX under the Type column.

For DDX datasets, the capacity reported is the maximum capacity of the dataset, not the current capacity. For MDX datasets, the current capacity is reported.

Jumbo Datasets

HowMessy automatically detects Jumbo datasets and adjusts its internal routines to read multiple files. The HowMessy report does not change for Jumbo datasets.

Chapter 2 - Accessing HowMessy

Introduction

The input to HowMessy is any database, the output can be either a standard 132-column report, or 175-column and 223-column reports using PCL output to either your attached printer via record mode or to an output file named Loadrept. The program file that you :Run is always HowMessy.Pub.Robelle.

Log on as the database creator in the same account as the database:

```
:hello dba.prod                          {must be CREATOR}
```

Create a :File command for the output file Loadrept. If you leave it out, HowMessy prints the report on $Stdlist.

```
:file loadrept;cctl;dev=lp,,copies

:build report;rec=-132,,f,ascii;cctl
:file loadrept=report,old;dev=disc
```

Run the HowMessy program:

```
:run howmessy.pub.robelle
```

Enter the database name.

```
Enter Database: SALE.DB
```

HowMessy attempts to open the database in mode-5 (read-only; allow concurrent updates by users who open in mode-1). If that fails, HowMessy tries mode-6 (read-only, shared; allow one update user who opens the database in mode-4). This means that you do NOT have to wait until you have exclusive access to the database to run HowMessy. Since HowMessy does not read down the chains of your database, it does not matter that people may be modifying the database while you are analyzing it. The only result is that the entry counts might be off by one or two.

After analyzing the database and printing a report, HowMessy continues to prompt for database names until you enter a blank line.

Converting DBLOADNG Job Streams

1. Add :File for Loadrept file.
2. Delete open-mode lines and dataset names. HowMessy reports on all datasets in a database, unless run with ;info="p".

3. Be sure to leave a blank line after the databases which you want to be reported.
4. Sample stream:

```
:JOB HOWMESSY,DBA.PROD,BASES
:COMMENT DISCARD $STDLIST IF NO ERRORS.
:FILE LOADREPT;DEV=LP;CCTL
:RUN HowMessy.Pub.ROBELLE
SALEDB
PURCH
AR

:EOJ
```

Partial Database Reports

You can report on individual datasets by running HowMessy with Info="P" (you can use lowercase or uppercase). This forces HowMessy to prompt for one or more dataset names to report. **Note:** if you select a detail dataset, HowMessy must serially read the master datasets for all paths in the detail dataset. In this case, HowMessy will produce a report for all the datasets that it has read.

```
:run howmessy.pub.robelle;info="p"
Enter Database: SALE.DB
Enter Dataset:  SALES
Enter Dataset:  CUSTOMERS
Enter Dataset:
```

HowMessy continues to prompt for dataset names until you enter a blank line. HowMessy does not analyze the requested datasets or print a report until you have entered the blank line.

PCL Reports

You can produce reports that utilize PCL output to output to your attached LaserJet printer via record mode or to a LaserJet attached to your HP3000.

Currently you can invoke the pcl option of the report by entering some commands in the info string.

The valid commands are:

Command	Purpose
R1	Record mode with pcl 1
R4	Record mode with pcl 4
R6	Record mode with pcl 6
W1	Loadrept with pcl 1
W4	Loadrept with pcl 4
W6	Loadrept with pcl 6
D	Enable duplex mode
A	Enable A4 paper

The PCL codes will product reports in the following formats:

PCL Code	Orientation	Dimensions
1 Lineprinter	landscape	175 cols/60 lines
4 Lineprinter	portrait	132 cols/80 lines

6 Lineprinter legal	landscape	223 cols/60 lines

The report for the r4 and w4 reports are exactly the same as previous versions of the report. The pcl1 and pcl 6 versions of the report have had some changes and allow for larger values.

In order to produce a report to your attached printer you invoke the PCL report format in the following manner.

The following run command invokes HowMessy with a PCL 1, report written to your attached LaserJet in Duplex Mode, and A4 Paper.

```
:run howmessy.pub.robelle;info="r1da"
```

The following run command invokes HowMessy with a PCL 1, report written to a the Loadrept file in Duplex Mode, and A4 Paper.

```
:run howmessy.pub.robelle;info="w1da"
```

Duplex mode is useful for those LaserJets that allow printing on both sides.

The A4 Option enables reports to be printed on "A4" paper.

System JCW

HowMessy sets the system job control word JCW to a fatal state when HowMessy fails in a batch job.

SPDEPREFETCH JCW

HowMessy can read data directly from disc into memory using Multi-Rec Nobuf reads. However, HowMessy is often slowed down on MPE/iX while waiting for the file system to satisfy its read requests. Using prefetch on MPE/iX, HowMessy is able to increase its throughput by instructing MPE/iX's memory manager to read the data from disc to memory ahead of time. This way, when HowMessy needs the data it is already in memory and HowMessy doesn't have to wait.

The SPDEPREFETCH JCW tells the memory manager how far ahead of HowMessy it should fetch data. Setting this number too low won't give the memory manager sufficient time to get the data into memory before HowMessy needs it. Setting the number too high may mean that on a busy system the data will be overwritten by something else before HowMessy gets a chance to use it.

When reading an input dataset, HowMessy/iX prefetches twice the buffer length of data from disc to memory. This default value of 2 works well, even on small machines. If HowMessy is running stand-alone on a fast CPU with a lot of memory, you may wish to increase the prefetch amount. The maximum value for SPDEPREFETCH is 5 (i.e., five times the buffer length). If you don't want HowMessy to prefetch, you can specify SPDEPREFETCH 0. This may be required when HowMessy is operating with third-party software tools that intercept all file system calls (e.g., Netbase from Quest Software).

The SPDEPREFETCH JCW is ignored by HowMessy/V.

Chapter 3 - Applying HowMessy

Introduction

To demonstrate the power of HowMessy, we will analyze an actual report on the internal efficiency of a database.

A Sample HowMessy Report

```
HowMessy/XL (Version 2.8)                                 Data Base: CUSTMR                    Run on: SUN, FEB 13, 1994,  4:52 PM
 for IMAGE/3000 databases                         By Robelle Solutions Technology Inc.                                  Page:    1
                                              Secon-  Max
                  Type                  Load  daries Blks  Blk                   Max     Ave     Std    Expd     Avg Ineff  Elong-
   Data Set           Capacity Entries Factor (Highwater) Fact    Search Field Chain   Chain     Dev  Blocks  Blocks Ptrs    ation
MCUST              Ato   21881   17295  79.0%  31.1%    6  28  CUSTOMER            6    1.45    0.71    1.08    1.08  6.1%    1.08
MSTATE             Man   18401   14722  80.0%  31.4%   15  11  ACCOUNT             7    1.46    0.72    1.00    1.31 23.5%    1.31
DNAME              Det   21876   17295  79.1% (    18336)   4 !CUSTOMER            1    1.00    0.00    1.00    1.00  0.0%    1.00
                                                             S ACCOUNT          245    1.17    3.14    1.03    1.17 53.3%    1.14
DRECV              Det  208196  118609  57.0% (   155612)  23S!ACCOUNT         1604    8.06   35.75    1.36   11.32 72.5%    8.34
```

The output of HowMessy is a file named Loadrept, and by default (as above) is formatted with 132 columns for fifteen-inch wide paper. With the PCL 1 and 6 options, 175 columns and 223 columns are available on the report. These versions of the report currently allow for larger values to be printed and indicates the Search Field data type. So if you have a particularly large database or are getting all 9's for a particular column we recommend that you use the PCL 1 or 6 options.

Explanation of HowMessy Columns

The sample above shows the default columns on the report, and each of them is described below:

V or XL

Indicator of which version produced the report.

If the indicator next to the word "HowMessy" is "/V", the compatibility-mode version produced the report. If the indicator is "/XL", the native-mode version produced the report.

Data Base

Name of the database as read in by HowMessy.

Run On

Date and time the report was run.

Data Set

Name of the dataset.

```
Data Set
MCUST
```

Datasets appear in the order that they are numbered by IMAGE (i.e., data set #1 is first, followed by data set #2). This is the same as their order in the schema. Master datasets always appear before the detail datasets to which they link.

Type

Type of the dataset.

```
Type
 Ato
```

In the Type column, "Ato" means Automatic Master Dataset, "Man" means manual master dataset and "Det" means detail dataset. If a dataset has dynamic expansion enabled, the symbol shown is ADX for automatic master datasets, MDX for manual master datasets, or DDX for detail datasets.

Capacity

Number of records the dataset can hold.

```
Capacity
   21881
```

For master datasets, whether automatic or manual, the *Capacity* should **not** be a power of two (e.g., not 2, 4 ... 1024, 2048 ... 65536 ...). For detail datasets, the capacity should be large enough to avoid overflowing before you have time to expand the dataset; IMAGE rounds the capacity of detail datasets up to the next even block.

If dynamic dataset expansion has been enabled, the capacity reported by HowMessy is the maximum capacity (for details) or the current capacity (for masters).

Entries

Number of entries currently in the dataset.

```
Entries
  17295
```

In detail datasets, the number of *Entries* is only significant if the dataset is about to overflow. In master datasets, the closer the number of entries is to the capacity, the slower additions to and retrievals from the dataset will be.

Load Factor

Percentage of the capacity currently in use.

```
 Load
Factor
 79.0%
```

Load Factor is calculated as "entries/capacity times 100". The performance of master datasets tends to degrade as the load factor approaches 100%. The performance of detail datasets is not related to load factor.

Secondaries

For master datasets only, percentage of "secondary" entries.

```
Secon-
daries
 31.1%
```

HowMessy calculates this column as "secondaries/entries times 100". IMAGE links primary entries (those that reside in their intended "hash location") to secondary entries (those that cannot reside in their hash location because the primary entry already resides there) via a forward- and backward-linked list for each primary entry. When searching for (or maintaining) entries, IMAGE hashes to the primary location and uses the pointers (if necessary) to find the desired record.

In general, the lower the percentage of secondaries in a master dataset, the better. However, in special cases there **can** be a high percentage of secondaries without a performance penalty (see *Ineff Ptrs*); there can also be serious problems **without a single secondary**.

Max Blks

For master datasets only, worst case of contiguous full blocks.

```
 Max
Blks
   6
```

The *Max Blks* column reveals locally bad hashing ("clustering"), which might be hidden in the averages. The higher this number, the longer a DBPUT into the "cluster" will take (because the primary is occupied, IMAGE must search serially forward through the cluster until it finds a free entry). The value shown in this column is the worst-case number of contiguous full blocks without a single free entry (the dataset is treated as a continuous loop, in that the last block loops around to the first block). The number in this column should be small (1-10, depending upon the blockfactor) if the key values are hashing in a randomly dispersed manner.

When IMAGE does a DBPUT, it hashes to the primary location. If the primary location is already occupied by another key value that hashed to the same location (a synonym), IMAGE searches sequentially forward in the dataset until it finds a free entry. This free entry is allocated to the new record and is linked to the forward/backward synonym list. The more blocks IMAGE must search for a free entry, the longer the DBPUT will take. For a "binary" search field (i.e., J1, J2, K1, K2, etc.) that is assigned consecutive key values (i.e., order numbers, etc.), a particularly horrendous situation can arise.

Assume that there are 40,000 entries in a master dataset with a capacity of 50,021 (not a power of 2). The currently assigned key values are 1 to 20,000 and 30,000 to 50,000. Each key value will hash perfectly (there are no synonyms), since IMAGE hashes by dividing the key value by the capacity and adding 1. Assume that key values are assigned sequentially, and that the next one is 50,001. What happens when the key value goes over 50,021 (i.e., the capacity)?

The first key value large enough to "wrap around" and come up with a hash location that is already occupied is going to cause IMAGE a big headache. IMAGE will search the vicinity of the primary location for a free entry and, finding none, will read serially through the dataset until it does. This could take up to *Max Blks* disc reads, effectively locking the computer.

Highwater

For detail datasets only, the highwater mark.

```
(Highwater)
(    18336)
```

For detail datasets, the record number of the *Highwater* mark is shown in parentheses. This number is useful, since serial reads must read to the highwater mark and not to the number of entries in the detail dataset.

Blk Fact

Number of records per physical block.

```
 Blk
Fact
 28
```

A low blockfactor sometimes explains bad results in the *Ineff Ptrs* and *Max Blks* columns. The blockfactor is determined by the value for $CONTROL BLOCKMAX in the schema file (default is 512 words, usually a reasonable choice), but IMAGE will sometimes select a blockfactor that is lower than necessary in order to save a few sectors of disc space.

You can specify an explicit blockfactor for an IMAGE dataset by placing it in parentheses after the capacity in the schema (but before the semicolon).

"S"

For detail datasets only, sorted path.

```
   Search Field
S ACCOUNT
S!ACCOUNT
```

An "S" appears if this is a *sorted path* into a detail dataset. When an entry is added to a sorted chain, IMAGE must search up the chain to find the correct position to insert the new entry. The longer the chain is, the slower this will be (see *Ave Chain* column), unless the entries are always added in sorted order.

"!"

For detail datasets only, primary path.

```
   Search Field
 !CUSTOMER
S!ACCOUNT
```

"!" appears if this is the *primary path* for a detail dataset. The primary path is the one used by DBUNLOAD to copy the entries from the dataset to the tape (unless the SERIAL option is used). After a DBLOAD, entries with the same value for the primary-path item will tend to be in the same physical block (i.e., they will be contiguous in the dataset). Their *Elongation* will be 1.00 (see last column of the HowMessy report).

What this means is that you can retrieve the entries on the primary path faster because they tend to be in the same disc block. For the primary path, you should select the path with the longest average chain length (greater than 1.00!), but also with a high frequency of access by on-line users. There is no point in optimizing a path that is seldom used. Of course, there is probably no point in having a path that is seldom used.

Search Field

Name of the search field and type.

```
   Search Field
  CUSTOMER
```

For master datasets, this is the single *Search Field* for the dataset. Detail datasets may have more than one search field, so HowMessy prints a separate line of analysis for each path into the dataset.

The Type of the Search Field will appear on the reports run with PCL 1 or PCL 6.

IMAGE uses two radically different hashing algorithms for search fields, depending upon the data type. Search fields of "Ascii" type (U, X, Z or P) are hashed using all bytes of the key value in the calculation, but search fields of "binary" type (I, J, K or R) are hashed using only the rightmost 31 bits of the search field. The last step of the hashing algorithm involves dividing by the capacity and saving the remainder. This suggests that "binary" keys should never be longer than four bytes and that you may still get very ugly results for certain patterns of key values. You may want to consider changing the type of the key value to Numeric Display (Z) in order to use the other hashing algorithm.

Max Chain

Number of records on the longest chain.

```
 Max
Chain
   6
```

For a master dataset, the length shown is for the maximum synonym chain. The shorter the synonym chains, the better for performance. However, if the *Max Chain* value is 1 for a master dataset and the *Load Factor* is greater than 50%, it means that there have been **no hashing collisions** at all. This is suspicious, indicating non-random hashing and potential problems (see *Max Blks*).

For detail datasets, this column shows the maximum number of entries with the same value for this search field. If you know that there should be only one detail entry per key value for this path, a value greater than 1 suggests inadequate checking in the application programs. If there are more than 99,999 records for a single key value, the maximum chain will be shown as 99999. You can try using the PCL 1 or 6 options, which allow larger values to be printed in all columns.

Ave Chain

Average length of a chain.

```
  Ave
 Chain
  1.45
```

The *Ave Chain* column is calculated as the total number of entries in the dataset, divided by the number of chains. For master datasets, this number is related to the *Secondaries* column and should be from 1.01 to 1.50. That is, the average synonym chain length should be short. For detail datasets, this number is application-dependent. In fact, you can use this number to optimize application programs (allow table space for the average number of line items on an order).

If the *Ave Chain* value is very large for a detail path, the path may not be necessary. A search path is an added structure for your database, which must be "paid for" through increased overhead. If the search path does not "divide" the dataset into a large number of small subsets, such as orders for a customer, the path is not doing the job of a path. An example of such a path would be the division number of an employee detail dataset, where there are only three unique division values. It would be faster to scan the dataset sequentially and select the employees with the desired division than to do a DBFIND and read down a chain that contains 1/3 of the entries in the dataset. This is especially the case if you have Suprtool, which can do serial selections much faster than QUERY or an application program.

Std Dev

Standard deviation of the chain length.

```
  Std
  Dev
 0.71
```

The *Std Dev* column shows the deviation from the average that is required to encompass 90% of the cases. If *Std Dev* is small, most of the chains are close to the *Ave Chain* length. If *Std Dev* is large, a significant number of chains are longer or shorter than the average (90% of the chains are within *Std Dev* of the average, plus or minus).

Expd Blocks

Average expected blocks per chain.

```
  Expd
 Blocks
  1.08
```

This column shows the average number of blocks each chain would occupy if entries with the same key value were adjacent. This value is computed as the sum of the expected blocks for each chain divided by the number of existing chains. *Expd Blocks* is used with *Avg Blocks* to calculate *Elongation*.

Avg Blocks

Average actual blocks per chain.

```
  Avg
 Blocks
  1.08
```

This column shows the actual average blocks per chain. This is computed as the sum of the pointers to different blocks, divided by the number of chains. The actual number of blocks used per chain may be several times the theoretically possible number of blocks; the number of blocks per chain determines the number of disc reads required to traverse the chain.

Ineff Ptrs

Percentage of pointers across block boundaries.

```
Ineff
 Ptrs
  6.1%
```

The *Ineff Ptrs* column is one of the key statistics produced by HowMessy. For master datasets, "inefficient pointers" is computed as the sum of the secondary pointers to different blocks, divided by the total number of secondary pointers. If the average chain length is small, this value may be skewed by a few records that end up in different blocks. For detail datasets, "inefficient pointers" is computed as the sum of the pointers to different blocks, divided by the total number of entries in the dataset.

If the pointer is to a record in the same disc block, no additional disc overhead is required to obtain the entry. If the pointer is to a record in another block, IMAGE must access that block as well as the current block. Obviously, it is desirable to minimize the percentage of inefficient pointers.

Elongation

Actual blocks divided by expected blocks.

```
Elong-
 ation
  1.08
```

This statistic, the ratio between actual and expected, tells how reality differs from the optimal case. An *Elongation* value of 1.00 is perfect. A value of 8.34 for a detail path may be terrible, if the chains of that path are accessed frequently. It means that to read an average chain takes 8 times more disc reads than if the entries were removed, sorted by the primary key value, then reloaded packed into adjacent entries. Note however that **only** the primary path can be optimized (unless you use Adager). You cannot pack all of the paths into a dataset.

Analyzing HowMessy Reports

To clarify how to use the HowMessy report, we will reproduce some actual results and analyze them. The first four datasets that we will study are from a customer database: MCUST, MSTATE, DNAME, DRECV. Then we will look at two examples from an inventory database: PRTBAL, POSTHIST.

MCUST Automatic Master

```
HowMessy/XL (Version 2.8)                              Data Base: CUSTMR                 Run on: SUN, FEB 13, 1994,  4:52 PM
 for IMAGE/3000 databases                      By Robelle Solutions Technology Inc.                              Page:    1
                                              Secon-  Max
                  Type                  Load  daries Blks  Blk                      Max     Ave      Std     Expd     Avg  Ineff  Elong-
   Data Set           Capacity Entries Factor (Highwater) Fact    Search Field    Chain   Chain      Dev   Blocks  Blocks Ptrs    ation
MCUST                Ato   21881   17295  79.0%  31.1%     6  28  CUSTOMER            6    1.45     0.71     1.08     1.08   6.1%   1.08
MSTATE               Man   18401   14722  80.0%  31.4%    15  11  ACCOUNT             7    1.46     0.72     1.00     1.31  23.5%   1.31
DNAME                Det   21876   17295  79.1% (     18336)   4 !CUSTOMER            1    1.00     0.00     1.00     1.00   0.0%   1.00
                                                               S ACCOUNT            245    1.17     3.14     1.03     1.17  53.3%   1.14
DRECV                Det  208196  118609  57.0% (    155612)  23S!ACCOUNT          1604    8.06    35.75     1.36    11.32  72.5%   8.34
```

As you can see, MCUST is an Automatic Master Dataset that is about 80% full. It comes from a customer maintenance database and provides inquiry via customer number into two detail datasets (required customer name and optional shipping address). Although this dataset is 80% full and has 31% secondaries (69% of the entries reside at their primary location), only 6% of the synonym pointers are "inefficient" (i.e., pointing out of the current block).

Thus, this dataset is actually quite efficient (only 31% of the entries are secondaries, and only 6% of the pointers to these secondaries cross a block boundary). The explanation for this efficiency is simple: the blockfactor is 28. Therefore, it takes a cluster of at least 28 records in one block before another entry hashing to that block will be given an inefficient synonym pointer.

We can also see that the longest synonym chain is only 6 entries, and the largest cluster of blocks without a free entry is only 6 blocks. This is about as good as an 80%-full master dataset can be expected to look.

MSTATE Manual Master

```
HowMessy/XL (Version 2.8)                               Data Base: CUSTMR                    Run on: SUN, FEB 13, 1994,  4:52 PM
 for IMAGE/3000 databases                       By Robelle Solutions Technology Inc.                                 Page:    1
                                              Secon-  Max
                  Type                   Load  daries Blks  Blk                    Max    Ave      Std    Expd    Avg  Ineff  Elong-
   Data Set           Capacity Entries Factor (Highwater) Fact    Search Field  Chain  Chain      Dev  Blocks  Blocks Ptrs    ation
MCUST             Ato    21881   17295  79.0%  31.1%     6   28  CUSTOMER           6    1.45     0.71    1.08    1.08   6.1%   1.08
MSTATE            Man    18401   14722  80.0%  31.4%    15   11  ACCOUNT            7    1.46     0.72    1.00    1.31  23.5%   1.31
DNAME             Det    21876   17295  79.1% (    18336)    4 !CUSTOMER           1    1.00     0.00    1.00    1.00   0.0%   1.00
                                                              S ACCOUNT          245    1.17     3.14    1.03    1.17  53.3%   1.14
DRECV             Det   208196  118609  57.0% (   155612)   23S!ACCOUNT         1604    8.06    35.75    1.36   11.32  72.5%   8.34
```

MSTATE is another master dataset from the same database, but this one is a Manual Master (it contains data fields, as well as the search field). The search field for this dataset is drawn from the same "domain" of customer numbers as the search field of the MCUST dataset, but it contains only "head offices" (they receive statements), rather than divisions of other head offices. Since the dataset is about the same percentage full as MCUST and has about the same key values, we would expect it to produce similar results in all columns.

However, there are two significant differences: 1) the *Ineff Ptrs* column is 23.5% (instead of 6.1%) and 2) the *Max Blks* column is 15 (instead of 6). This dataset is less efficient than the previous one, even though it has the same percentage of *Secondaries*, the same *Max Chain* length, *Ave Chain* length, and *Std Dev*. The reason is easy to see: MSTATE has a blockfactor of only 11 records per block, as opposed to the 28 per block in MCUST.

DNAME Detail Dataset

```
HowMessy/XL (Version 2.8)                               Data Base: CUSTMR                    Run on: SUN, FEB 13, 1994,  4:52 PM
 for IMAGE/3000 databases                       By Robelle Solutions Technology Inc.                                 Page:    1
                                              Secon-  Max
                  Type                   Load  daries Blks  Blk                    Max    Ave      Std    Expd    Avg  Ineff  Elong-
   Data Set           Capacity Entries Factor (Highwater) Fact    Search Field  Chain  Chain      Dev  Blocks  Blocks Ptrs    ation
MCUST             Ato    21881   17295  79.0%  31.1%     6   28  CUSTOMER           6    1.45     0.71    1.08    1.08   6.1%   1.08
MSTATE            Man    18401   14722  80.0%  31.4%    15   11  ACCOUNT            7    1.46     0.72    1.00    1.31  23.5%   1.31
DNAME             Det    21876   17295  79.1% (    18336)    4 !CUSTOMER           1    1.00     0.00    1.00    1.00   0.0%   1.00
                                                              S ACCOUNT          245    1.17     3.14    1.03    1.17  53.3%   1.14
DRECV             Det   208196  118609  57.0% (   155612)   23S!ACCOUNT         1604    8.06    35.75    1.36   11.32  72.5%   8.34
```

DNAME is a Detail Dataset with two search fields. Notice that it has exactly the same number of entries as MCUST, which is good because each customer number is supposed to have one (and only one) DNAME entry associated with it (one name and address per customer number). The *Highwater* is 18336. Serial reads of this dataset will have to read 18336 records (you might have expected IMAGE to read 17295 entries). There is room for 1041 new entries (assuming no deletes) before the highwater mark must be increased.

The first *Search Field* is CUSTOMER, linked to the MCUST dataset. The *Max Chain* length is 1 (only one address per customer number), the *Ave Chain* length is 1 (at least one address per customer number), the *Std Dev* is 0 (there are no exceptions), and there are no *Ineff Ptrs* (because there are no pointers in a chain of length 1).

The second *Search Field* is ACCOUNT, linked to the MSTATE dataset. This field contains the customer number of the head office. For most DNAME entries, the ACCOUNT field equals the CUSTOMER field. We can deduce this because the *Ave Chain* length is only 1.13. A few customers must be very large, however, since the *Max Chain* value is 245 (one customer has 245 divisions reporting to one head office) and the *Std Dev* is 3.14. There may not be many pointers on these chains (since the most common chain length is 1), but the chains which do exist are very inefficient (53.3% percent of the existing pointers point to different blocks).

There is one "mistake" in the way this dataset is set up: ACCOUNT should be the primary path, not CUSTOMER. CUSTOMER became the primary path by default; the first unsorted path in the dataset is the primary path if you do not specify one explicitly, and you cannot optimize a path whose chains are always 1 in length. ACCOUNT, on the other hand, has many chains of length greater than 1 (at least one chain has 245 members). If it were the primary path, you could reduce the percentage of *Ineff Ptrs* by unloading and reloading the dataset.

DRECV Detail Dataset

```
HowMessy/XL (Version 2.8)                                 Data Base: CUSTMR                    Run on: SUN, FEB 13, 1994,  4:52 PM
 for IMAGE/3000 databases                         By Robelle Solutions Technology Inc.                                  Page:    1
                                              Secon-  Max
                  Type                  Load  daries Blks  Blk                   Max    Ave      Std    Expd    Avg  Ineff  Elong-
   Data Set           Capacity Entries Factor (Highwater) Fact    Search Field  Chain  Chain     Dev   Blocks Blocks Ptrs    ation
MCUST              Ato   21881   17295  79.0%  31.1%    6  28  CUSTOMER             6   1.45    0.71     1.08   1.08  6.1%    1.08
MSTATE             Man   18401   14722  80.0%  31.4%   15  11  ACCOUNT              7   1.46    0.72     1.00   1.31 23.5%    1.31
DNAME              Det   21876   17295  79.1% (    18336)   4 !CUSTOMER             1   1.00    0.00     1.00   1.00  0.0%    1.00
                                                             S ACCOUNT            245   1.17    3.14     1.03   1.17 53.3%    1.14
DRECV              Det  208196  118609  57.0% (   155612)  23S!ACCOUNT           1604   8.06   35.75     1.36  11.32 72.5%    8.34
```

DRECV is a detail dataset which holds Accounts Receivable entries, indexed by ACCOUNT (customer number of the head office) from the MSTATE master dataset. The HowMessy report shows that the average customer account has 8.06 A/R transactions (invoices, payments, etc.), but the largest account has 1604 transactions and the *Std Dev* is 35.75 entries (there are a substantial number of accounts with three or four times the average).

Why would a path with chains of up to 1604 entries be **sorted**? The fact that the path is sorted does not cause an efficiency problem in this case because the sort field is the DATE, and entries are usually added in date order anyway. When a user does an inquiry into the Accounts Receivable for a customer, the entries are almost always required in date order, so the sort field makes sense.

This path is the first one with a large *Elongation* value (8.34). This high value indicates that the DRECV dataset could be made much more efficient by unloading it and reloading it (along the primary path). If entries with the same key value were loaded into adjacent locations in the dataset, the average chain would occupy only 1.36 physical blocks. As the dataset stands, the average chain actually traverses 11.32 blocks (72.5% of the pointers are inefficient). This situation arises naturally over time because the entries are loaded in date order, not in customer order (new invoices and payments are loaded every day). This dataset is 8 times more inefficient than it need be, but the improvement gain from a reload would only be temporary.

PRTBAL Detail Dataset

This example is a detail dataset called PRTBAL with three paths. PRTBAL stands for "part balance" and each entry records the quantity on hand for a specific part at a specific location:

```
HowMessy/XL (Version 2.8)                                 Data Base: INVENT                    Run on: SUN, FEB 13, 1994,  4:52 PM
 for IMAGE/3000 databases                         By Robelle Solutions Technology Inc.                                  Page:    1
                                              Secon-  Max
                  Type                  Load  daries Blks  Blk                   Max    Ave      Std    Expd    Avg  Ineff  Elong-
   Data Set           Capacity Entries Factor (Highwater) Fact    Search Field  Chain  Chain     Dev   Blocks Blocks Ptrs    ation
PRTBAL             Det   45001   32664  72.6% (    35245)  11 !PARTBASEKEY          2   1.00    0.00     1.00   1.00  6.0%    1.00
                                                               BASENUM          19709 379.80 2399.00  742.00 2163.00 27.0%    2.91
                                                               PARTNUM              4   1.31    2.80     1.10   1.34 22.0%    1.22
POSTHIST           Det   80003   56643  70.8% (    56745)   7 !INVCENUM          5611   7.04   63.00     1.63   2.91 27.0%    1.79
                                                               PARTNUM          40174   1.34  188.00     1.87   3.10 29.0%    1.66
                                                               PONUM            16445   1.82   93.00     3.22   6.57 31.0%    2.04
```

PARTBASEKEY is a composite key formed by combining the PARTNUM and the BASENUM (location). The *Ave Chain* length is 1, but the *Max Chain* length is 2.

Since there can be only one valid on-hand quantity for any part at any one location, the maximum chain length expected is 1. There must be a duplicate entry in this dataset. Since the maximum expected chain length is 1, there is no point in selecting this as the primary path.

PARTNUM is the inventory part number. The *Max Chain* length is 4 because some parts are stocked at all three inventory locations (there are three values for BASENUM) and there is at least one duplicate entry (as deduced in the previous paragraph).

BASENUM is a code identifying the location where the inventory is stocked. The *Max Chain* value of 19709 (out of 32664 total entries) is for the corporate headquarters, while the remaining inventory is either at the other major office or in transit. Does it make sense to have BASENUM as a search field in this dataset? There are 2921 data blocks in PRTBAL (32664 entries divided by a blockfactor of 11 equals 2921 blocks), while *Avg Blocks* per chain is 2163. We are not saving many disc accesses by having BASENUM as a key, versus using a serial search to select the same entries. This is a classic case of an unnecessary key, due to the small number of possible key values.

POSTHIST Detail Dataset

The second example from the inventory database is a detail dataset called POSTHIST, again with three paths. POSTHIST maintains relationships between invoices, purchase orders, and back orders for specific inventory parts:

```
HowMessy/XL (Version 2.8)                              Data Base: INVENT                    Run on: SUN, FEB 13, 1994,  4:52 PM
 for IMAGE/3000 databases                      By Robelle Solutions Technology Inc.                                 Page:    1
                                                   Secon-  Max
                  Type                     Load  daries Blks  Blk                      Max    Ave      Std     Expd     Avg  Ineff  Elong-
   Data Set           Capacity Entries Factor (Highwater) Fact    Search Field  Chain  Chain      Dev   Blocks  Blocks Ptrs   ation
PRTBAL             Det    45001    32664  72.6% (     35245)  11 !PARTBASEKEY        2   1.00     0.00     1.00    1.00   6.0%   1.00
                                                                  BASENUM        19709  379.80 2399.00   742.00 2163.00  27.0%   2.91
                                                                  PARTNUM            4   1.31     2.80     1.10    1.34  22.0%   1.22
POSTHIST           Det    80003    56643  70.8% (     56745)   7 !INVCENUM        5611   7.04    63.00     1.63    2.91  27.0%   1.79
                                                                  PARTNUM        40174   1.34   188.00     1.87    3.10  29.0%   1.66
                                                                  PONUM          16445   1.82    93.00     3.22    6.57  31.0%   2.04
```

Based on our experience with the PRTBAL dataset above, we might question the PARTNUM path here (since 40174 of the 56643 entries have the same value!) and recommend removing the path. In this case, we would be wrong. The reason is that PARTNUM is always blank in POSTHIST entries, unless the entry is a back order allocated to a specific part number (only 16356 out of 56643 entries are so allocated). The *Ave Chain* length is close to 1 (one back order per part) and quick retrieval is important in the 16356 cases that are back orders.

A search field such as PARTNUM in POSTHIST is a common method of isolating a minority of cases requiring special attention. The majority of entries in POSTHIST require no special handling, and thus are lumped together on the "default" chain (PARTNUM equals blanks). One problem with this design is that the default chain will soon exceed the maximum chain length allowed by IMAGE (65535 entries on a single chain without TurboIMAGE). When that happens, subsequent DBPUTs to the default chain will fail. One solution is to use a default value that varies (e.g., two blanks, followed by today's date). Another solution is to remove the path from this dataset and create a new detail dataset to provide a cross-reference.

Resolving Problems

There are different problems for master datasets and detail datasets. The actions you can take regarding the situations uncovered by HowMessy are determined by what software tools you have available.

Changing key types or their format is almost impossible after a large database is in production. Build your test databases early and fill them with an appropriate set of values (or build your entire application database early if you have the resources). Use HowMessy on your test database. Early database changes are always the easiest to implement.

Master Dataset Solutions

If secondaries are over 30% and inefficient pointers are over 50%, the dataset is either too full or not hashing properly. Increase capacity to a higher odd number, or change the data type and format of the search field.

Increasing block factor should reduce inefficient pointers.

Look for clustering problems: load factor less than 80%, secondaries less than 5%, and maximum blocks greater than 100. Clustering usually results from a sequential binary key; change it to type X, U, or Z.

Detail Dataset Solutions

Ignore load factor, unless dataset overflow is likely.

If the highwater is much larger than the entries, repack the dataset to reduce the highwater to the number of entries in the dataset. This will speed up serial reads. Other database changes may also change the highwater mark -- check with the specific vendors.

If a detail dataset has more than one path, check that the primary path (!) has a large average chain length and is often accessed. The primary path should not have an average chain of 1.00 and a maximum chain of 1. If this is the case, assign the primary path to another path that is frequently accessed.

Elongation tells how inefficiently packed the chains are, relative to their optimum packing. Elongation of eight on a primary path means that disc I/O will be reduced by a factor of 8 if you repack the dataset. Adding and deleting detail dataset records increases the elongation. Be prepared for periodic repacking to maintain maximum performance.

Look for average chain equal to 1.00, standard deviation about 0.01, and maximum chain of 2 or 3; this is usually a dataset that should have exactly one entry per key value but actually has duplicate key values.

Look for paths with long chains (average chain plus standard deviation > 100), especially if the path is sorted (S).

Using DBUNLOAD and DBLOAD

Change $CONTROL BLOCKMAX to make blocks bigger or smaller. Change capacity of datasets (increase capacity to reduce synonyms in masters, allow room for expansion in details). Repack each detail dataset so that entries with the same primary path value are contiguous, and to reset the highwater mark to the number of entries in each detail dataset. Select a different primary path for a detail dataset (but it takes two "unload/load" cycles to get the entries packed along a new primary path). Remove a sort field. Remove an unneeded search path. You cannot add or delete datasets, add or delete fields (except from the end of an entry), or change the data format of a key. DBUNLOAD and DBLOAD are a standard part of IMAGE.

Using Adager

Change capacity (DETCAP, MASTCAP). Remove unwanted paths (PATHDEL). Select a different primary path (PRIMARY). Remove a sort field (SORTDEL). Add or delete fields (FIELDADD, FIELDDEL). Add or delete datasets (SETADD, SETDEL). Repack a detail dataset (DETPACK). If the primary path is wrong, you can pack the dataset along the correct path. Adager is a software product of Adager (telephone: 208-726-9100).

Using Suprtool

Convert a manual master into an automatic master plus a new detail, and copy data into the new detail (easiest with Adager). Unload and reload a single dataset. Convert batch applications to use the high-speed serial scan of Suprtool, and remove the unneeded path from the dataset using DBUNLOAD/DBLOAD or Adager. Suprtool is a product of Robelle Solutions Technology Inc.

Using DBMGR

Repack the primary path of a detail dataset so that it requires the minimum disc accesses (this is very fast). Change capacity of master and detail datasets. Erase datasets quickly. DBMGR is a software product of D.I.S.C. (telephone: 303-444-4000).

Using DBGENERAL

Change capacity (function 3.3, 3.5). Remove unwanted paths (function 5.5). Select a different primary path (function 5.5). Remove a sort field (function 5.5). Add or delete fields (function 5.4). Add or delete datasets (function 5.3). Repack the primary path of a detail dataset (function 3.6). Erase datasets quickly (function 4.4). DBGENERAL is a software product of Bradmark Computer Systems (telephone: 713-621-2808).

Writing Custom Programs

Change the format of data fields, including search fields.

Chapter 4 - Self-Describing Loadfile

Introduction

When HowMessy is successfully run, it produces a temporary file called Loadfile. The Loadfile is self-describing so that it can be used immediately by Suprtool or any other tool that understands Robelle's advanced self-describing file structure (e.g., AskPlus). The Loadfile contains all of the information from the HowMessy report, but a file is better than a report for automated processing. HowMessy purges any existing temporary Loadfile.

Form of the Loadfile

The following is a Suprtool Form command of the Loadfile:

```
File: LOADFILE.GROUP.ACCT      (SD Version B.00.00)
   Entry:                     Offset
      DATABASE             X26     1
      DATASET              X16    27
      DATASETNUM           I1     43
      DATASETTYPE          X4     45
      CAPACITY             I2     49
      ENTRIES              I2     53
      LOADFACTOR           I2     57                << .2  >>
      SECONDARIES          I2     61                << .2  >>
      MAXBLOCKS            I2     65
      HIGHWATER            I2     69
      PATHSORT             X1     73
      PATHPRIMARY          X1     74
      BLOCKFACTOR          I1     75
      SEARCHFIELD          X16    77
      MAXCHAIN             I2     93
      AVECHAIN             I2     97                << .2  >>
      STDDEVIATION         I2    101                << .2  >>
      EXPECTEDBLOCKS       I2    105                << .2  >>
      AVERAGEBLOCKS        I2    109                << .2  >>
      INEFFICIENTPTRS      I2    113                << .2  >>
      ELONGATION           I2    117                << .2  >>
      CURRENTDATE          I2    121                <<YYYYMMDD>>
      CURRENTTIME          I1    125
      EXPANSION            I1    127
      FUTUREFIELDS         X128  129
Limit: 10000  EOF: 5  Entry Length: 256  Blocking: 35
```

Suprtool Processing

You can use Suprtool to automatically select Loadfile records based on any fields in the Loadfile. For example, we will select all detail datasets with a load factor greater than 85%:

```
:run suprtool.pub.robelle
>input   loadfile
>if      datasettype = "D" and loadfactor > 85.0
>sort    database
>sort    dataset
>dup     none,keys
>list
>xeq
```

COBOL Layout

The Loadfile can be read by user programs. A COBOL program could produce a custom HowMessy report or it could select specific datasets for further examination (just like we can with Suprtool). This is the COBOL declaration for the Loadfile:

```
01  loadfile-record.
    05  database                pic x(26).
    05  dataset                 pic x(16).
    05  datasetnum              pic s9(4) comp.
    05  datasettype             pic x(4).
    05  capacity                pic s9(9) comp.
    05  entries                 pic s9(9) comp.
    05  loadfactor              pic s9(7)v99 comp.
    05  secondaries             pic s9(7)v99 comp.
    05  maxblocks               pic s9(9) comp.
    05  highwater               pic s9(9) comp.
    05  pathsort                pic x.
    05  pathprimary             pic x.
    05  blockfactor             pic s9(4) comp.
    05  searchfield             pic x(16).
    05  maxchain                pic s9(9) comp.
    05  avechain                pic s9(7)v99 comp.
    05  stddeviation            pic s9(7)v99 comp.
    05  expectedblocks          pic s9(7)v99 comp.
    05  averageblocks           pic s9(7)v99 comp.
    05  inefficientptrs         pic s9(7)v99 comp.
    05  elongation              pic s9(7)v99 comp.
    05  currdate                pic s9(9) comp.
    05  currtime                pic s9(4) comp.
    05  expansion               pic s9(4) comp.
    05  futurefields            pic x(128).
```

Pascal Layout

The Pascal declaration for the Loadfile is compatible with both Pascal/V and Pascal/iX (even with the $HP3000_32$ option in effect). This is the Pascal declaration for the Loadfile:

```
shortint = -32768 .. +32767;   {remove for Pascal/iX}

loadfiletype =
   record
      dbname       : packed array[1..26] of char;
      dbset        : packed array[1..16] of char;
      dbnum        : shortint;
      dbtype       : packed array[1..4] of char;
      capacity     : integer;
      entries      : integer;
      loadfactor   : integer;
      secondaries  : integer;
      maxblks      : integer;
      highwater    : integer;
      pathtype     : packed array[1..2] of char;
      blkfact      : shortint;
      searchfield  : packed array [1..16] of char;
      maxchain     : integer;
      avechain     : integer;
      stddev       : integer;
      expblks      : integer;
      aveblks      : integer;
      ineffptrs    : integer;
      elong        : integer
      currdate     : integer;
      currtime     : shortint;
      expansion    : shortint;
   end;
```

C Layout

This is the C declaration for the Loadfile:

```
struct Loadfile
{
   char  database[26];
   char  dataset [16];
   short dbnum;
   char  dbtype[4];
   long  capacity;
   long  entries;
   long  loadfactor;
   long  secondaries;
   long  maxblocks;
   long  highwater;
   char  pathtype[2];
   short blockfactor;
   char  searchfield[16];
   long  maxchain;
   long  avgchain;
   long  stddev;
   long  expblocks;
   long  avgblocks;
   long  ineffptrs;
   long  elong ;
   long  currdate;
   short currtime;
   short expansion;
   char  futurefields[128];
};
```

Chapter 5 - Installing HowMessy

Introduction

There are three steps to installing the HowMessy program on your system.

1. Restore the files from the tape.
2. Upgrade the Robelle account.
3. Install the correct version of HowMessy.

For steps 1 and 2, log on as Manager.Sys. If you received HowMessy along with another Robelle product (e.g., Suprtool), you should follow the installation steps for that product first. Then proceed directly to step 3 of the HowMessy installation. You need to log on as Mgr.Robelle for step 3.

Step 1: Restore from the Tape

Restore all the files from the Robelle distribution tape:

```
:hello manager.sys
:file robtape;dev=tape
:restore *robtape; @.@.robelle; create {=reply on console}
```

Step 2: Upgrade the Robelle Account

Stream the job that sets up the Robelle account with the proper structure and capabilities:

```
:stream robelle.job.robelle
```

After the job is complete, apply (or re-apply) a password to the Robelle account:

```
:altacct robelle;pass=something-hard-to-guess
```

Step 3: Install the Program

Our Bonus.Job.Robelle job stream installs this new version of HowMessy. No one can be using HowMessy during installation. Warn people not to use these programs for a while, and then stream our installation job:

```
:hello mgr.robelle
:warn @;please stop using HowMessy NOW!

:stream bonus.job                          {supply passwords}
```

Check the installation job $STDLIST. If anyone was using HowMessy or the other files, or attempting to back them up, the job will fail. Chase away any users, ensure that backup is not in progress, then stream the installation job again.

After the installation job completes, you are ready to use this new version of HowMessy.

```
:run howmessy.pub.robelle
```

Documentation

The HowMessy documentation is available in Adobe Acrobat PDF format for easy printing, and in Windows HTML Help format for easy viewing on-screen. You will find these files on the Robelle website at http://www.robelle.com/library/manuals/.

Appendix A - Error Messages

Introduction

The following error messages are produced by HowMessy. The first three are likely to happen because of user errors in accessing HowMessy. The other errors are caused by an invalid database structure or an internal error in HowMessy.

Error-1: Unable to open the database in mode-5 or mode-6

HowMessy has attempted to open the database with DBOPEN, but this has failed. Two common reasons are that someone has the database open exclusively (e.g., mode-3) or the database name was not spelled correctly.

Error-2: You are not the database creator

HowMessy was not run by the database creator. Check that you are logged on as the database creator in the same group as the database.

Error-3: Database name too long

Do not specify the group or account of the database.

Error-4: Dataset name too long

The dataset name specified does not exist in the database or it contains invalid characters for a dataset name.

Error-5: Invalid dataset name

A dataset name longer than 16 characters was specified.

Error-6: Unable to open the Loadrept file

HowMessy was unable to open the Loadrept file. Check that there are no unexpected file commands for the file Loadrept.

Error-7: Unable to write to the Loadrept file

HowMessy attempted to write a record to the Loadrept file, but the write failed. If the Loadrept file has been redirected to a disc file, ensure that the disc file has not filled up.

Error-8: Unable to close the Loadrept file

HowMessy could not close the Loadrept file. If the Loadrept file has been directed to a disc file, check for duplicate file names.

Error-9: Unable to purge Loadfile

To purge any existing Loadfile, HowMessy executes the MPE command:

```
:purge loadfile,temp
```

When this error appears, the MPE error message is also shown.

Error-10: Unable to open Loadfile

HowMessy was unable to open the Loadfile file. Check that there are no unexpected file commands for the file Loadfile.

Error-11: Unable to close Loadfile

HowMessy could not close the Loadfile file. If the Loadfile file has been directed to a different filename, check for duplicate file names.

Error-12: Unable to write labels to Loadfile

This error should never happen.

Error-13: Speed Demon error

HowMessy uses Speed Demon for access to each database. If you are a Suprtool user, please see the Speed Demon User Manual for a description of any Speed Demon errors. If you do not use Suprtool, contact Robelle.

Error-14: Unable to obtain current date

HowMessy was unable to get the current date from the system.

Error-15: Invalid PCL number. Only 1,4 or 6 are valid

An invalid PCL number was specified.

Glossary of Terms

capacity

The maximum number of entries a dataset may contain.

database

A collection of data files (detail datasets) and lookup files (master datasets) , organized and described by a root file, and accessed with a set of callable intrinsics.

dataset

An area of storage in a database. IMAGE allows a maximum of 199 datasets. A dataset has an entry format and a structure. There are two basic structures: master and detail datasets. IMAGE datasets are privileged files.

detail dataset

A dataset whose entries can have zero, one, or more than one search field. More than one entry can have the same value for a search field. An entry's position within a detail dataset is not related to the value of the search field.

master dataset

A dataset with one search field, and with at most one entry per search field value. A master dataset provides fast access by search field values. There are two types of master datasets: automatic and manual.

path

The structural relationship between a master dataset and a detail dataset. A path relationship causes IMAGE to maintain chains of entries having the same search field value. Paths may be sorted.

primary

Short for primary entry. An entry in a master dataset which is residing at the primary address calculated for it by the hashing function.

primary address

The record number of an entry in a master dataset as derived from the value of the entry's search field.

primary path

The detail path that will be optimized by database management tools. The default primary path is the first unsorted path in the schema. A primary path is not the same as a primary.

secondary

Short for secondary entry. A master entry that does not reside at the primary address calculated for it.

sort field

A field in a detail dataset (not the search field) whose value is used to sort the chain of the search field in ascending order.

Index