Paper 283-2010

# Using IBM Tivoli Storage Manager to Manage Large Datasets at IMS (and other data management tips)

Paul Doucette, IMS, Plymouth Meeting, PA

## ABSTRACT

During the course of a typical month, the LifeLink™ Statistical Services Programming Group at IMS processes 30-50 projects using datasets averaging 50 gigabytes in size (per dataset). A key tool that enables the cost-effective management of this volume of data is IBM's Tivoli Storage Manager (TSM) for Space Management. The use of TSM will be examined here as will other data management techniques.

## INTRODUCTION

The LifeLink programming group provides programming services in support of IMS' Consulting organization as they deliver customized market research analyses to pharmaceutical and biotech clients using Anonymized Patient Level Data (APLD)/Longitudinal Prescription Data (LRx). IMS' LRx database contains approximately 64% of all retail prescriptions dispensed in the U.S. so it's not unusual for the datasets processed by the LifeLink group to exceed 100 gigabytes in size and contain hundreds of millions of observations. In fact, as our database continues to grow and our clients' desire to analyze historical data increases, we're beginning to see regular occurrences of datasets exceeding 1-2 billion observations! Table 1.1 shows maximum dataset sizes for a typical sampling of 10 projects.

At any given time, there are approximately 20-25 custom deliverables being programmed and analyzed which follow a project timeline of approximately 3-6 weeks per project. The group is also responsible for executing approximately 50 recurring deliverables which follow monthly, quarterly, semi-annual or annual cycles. Considering the number of projects and the size of the datasets being processed, one can see why a tool like IBM's TSM is necessary. This paper will provide a basic overview of how TSM is used at IMS to support these large data volumes and will secondarily examine other data management efficiencies in use by the LifeLink programming group

**Table 1.1 Sample Project Data set Sizes**

| | Filesize |
|---|---|
| Project1 | 11.6 Gb |
| Project2 | 54 Gb |
| Project3 | 4.5 Gb |
| Project4 | 8.0 Gb |
| Project5 | 44 Gb |
| Project6 | 86 Gb |
| Project7 | 51 Gb |
| Project8 | 23 Gb |
| Project9 | 13 Gb |
| Project10 | 6.2 Gb |
| Total | 301.3 Gb |

## PROGRAMMING ENVIRONMENT OVERVIEW

The programmers in the LifeLink programming primarily work from connections to remote HP blade workstations with SAS® 9.2 which connect remotely to a Unix server running SAS 9.1.3. The data used by the group is extracted from a production database and flat files are delivered to the group's Unix box. Once the data files is extracted and on the server, the project programmer will read in the flat file, subsetting it appropriately, to create the initial SAS dataset. After that, the programmers conduct a series of data manipulation steps to create intermediate (analysis) datasets and also perform descriptive and statistical analyses. The programmers work closely with methodologists (statisticians) during this time with the methodologists directing the programmers in the proper data manipulations and analyses needed for the project. The programming group's final deliverable to the consulting team is usually a flat file or an Excel spreadsheet they then use to generate a final client deliverable. The final deliverable is typically in the form of a PowerPoint presentation, an Excel Pivot table, a database/data cube or some combination thereof.

The goal of these client deliverables is to inform business decisions related to the marketing of new and/or established pharmaceutical products. Clients are also looking at trends in prescribing and patient usage and are therefore quite interested in historical views of the data. Typical study periods include a reporting period of one year, a "lookback" period of one year and a "look forward" period of 3 to 6 months. In all, a typical study will need to process 2+ years of data. As products continue to mature, clients are more and more interested in historical data which can extend back 3-4 years (and sometimes longer).

The bottom line being that the more data we have, the more data clients want to analyze. In addition, there are several other reasons for needing to maintain a large inventory of data:

- the nature of our consulting projects is such that they prompt follow-up questions and additional analyses to provide greater insights, which means we need to have the data readily available for additional analyses (and they always need it yesterday!).
- projects can often take advantage of the data used by previous projects which reduces the need to extract data from the production database (which is constrained by production delivery cycles and other factors).
- projects executed on a recurring basis (e.g. monthly) can utilize data from prior runs.

## TSM BASICS

The use of IBM's Tivoli Storage Manager for Space Management (TSM) hierarchical storage management (HSM) product has essentially turned about 6* Tb of physical disk space into 293* Tb of "virtual" (tape-based) storage space, and is limited only by how much tape storage we are willing to invest in and how much room there is on the tape rack. Table 1.2 shows a recent snapshot of usage statistics for the fileshares available on our system.

**Table 1.2 Unix vs. TSM Storage Statistics**

| | ----------- Unix --------- | | | | ---------- HSM ------------- | | |
|---|---|---|---|---|---|---|---|
| **Filesystem** | **Size** | **Occupancy** | **%Used** | **Ratio** | **Size** | **Occupancy** | **%Used** |
| **/fs33** | 254 Gb | 142 Gb | 57 | 37 | 9486 Gb | 242 Gb | 2 |
| **/fs34** | 478 Gb | 379 Gb | 80 | 81 | 38862 Gb | 24001 Gb | 61 |
| **/fs35** | 127 Gb | 64 Gb | 51 | 49 | 6278 Gb | 2121 Gb | 33 |
| **/fs36** | 364 Gb | 257 Gb | 71 | 71 | 26086 Gb | 19448 Gb | 74 |
| **/fs37** | 180 Gb | 120 Gb | 67 | 44 | 8057 Gb | 2402 Gb | 29 |
| **/fs440.1** | 1006 Gb | 752 Gb | 75 | 137 | 138105 Gb | 100756 Gb | 72 |
| **/fs501.1** | 1003 Gb | 656 Gb | 66 | 27 | 27686 Gb | 19285 Gb | 69 |
| **/fs559.1** | 1959 Gb | 1640 Gb | 84 | 7 | 15360 Gb | 10152 Gb | 66 |
| **/fs583.1** | 1023 Gb | 136 Gb | 14 | 20 | 20470 Gb | 136 Gb | 0 |
| **/fs672.1** | 695 Gb | 524 Gb | 76 | 34 | 23942 Gb | 10950 Gb | 45 |

* this is up from 180 Tb in early 2009; in the above table /fs583.1 is not being managed by TSM so is not included in the totals

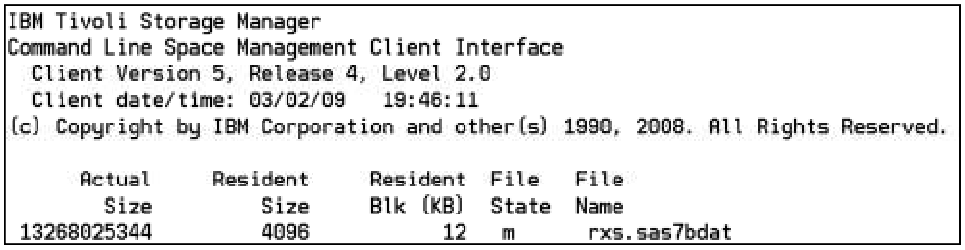
TSM accomplishes this through software which uses size and update/access times to determine which files to move and when to move them. Migrated files are stored in the data center using state of the art tape storage technology. The details of how TSM works is beyond the scope of this paper but the IBM website provides extensive detail on how TSM works. A link to the TSM site is provided at the end of the paper. Basically, the idea is this: TSM constantly scans the fileshares for "large" files that haven't been accessed or modified in a predetermined amount of time and moves those files to tape. Before doing so, though, the file first goes into a "pre-migrated" state where there are two copies of the file – one resident within the disk-based portion of the file system and a second copy on tape. This makes the actual move to tape (i.e. migrating) and restoration that much quicker. After TSM sees that a file has been in a pre-migrated state and has not been modified for a certain period of time, the file will then be migrated to tape, with an identifier (or stub or metadata) left behind. Table 1.3 below shows the various stages of file status. For disaster recovery purposes, IMS always maintains two copies of the tape – one kept on-site and one kept off-site. Further, an additional two copies of HSM-managed files are maintained on tape (one in the library and a second off-site) since HSM copies on tape are not subject to the same retention rules as are backup copies.

The dsmls command, which is the TSM version of the Unix ls (or list) command, displays the file state for all files following the list command. This command is demonstrated in Table 1.3. In the first screenshot, note the difference between the Resident Size and the Actual Size. This indicates that only the "stub" is resident within the disk based portion of the filesystem while the actual file (all 13+ Gb of it) has been migrated to tape. This is confirmed by the "m" in the File State column, which indicates that the file has been migrated to tape. In the second screenshot, the dsmrecall command was issued resulting in the file being recalled from tape and moved into a pre-migrated state as denoted by the "p" in the File State column. Note the resident and actual sizes are now the same. This recall of a 13 Gb file took less than 5 minutes, which is about average. The recall time is dependent on a number of factors including the size of the file, the number of files being recalled at that particular time, the availability of tape drives not occupied by other HSM or backup operations at the time, perhaps a queue of other backup and/or HSM tape operations that must be processed first, as well as any other processing being done on the SAS server at the time the recall is issued. Usually, the recall time is around the 5 minute range. If it goes beyond 10 or 15 minutes, we know there's a problem and will give our Unix/TSM Admin team a call.

The last screenshot in Table 1.3 simply shows files in all 3 file states: resident, pre-migrated and migrated.

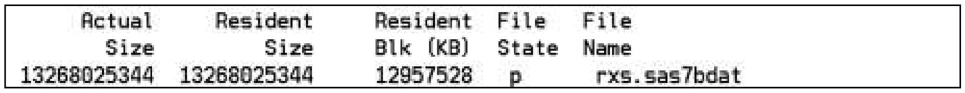
**Table 1.3 TSM Storage Manager File Status**

Results of dsmls command showing a file in a migrated state:



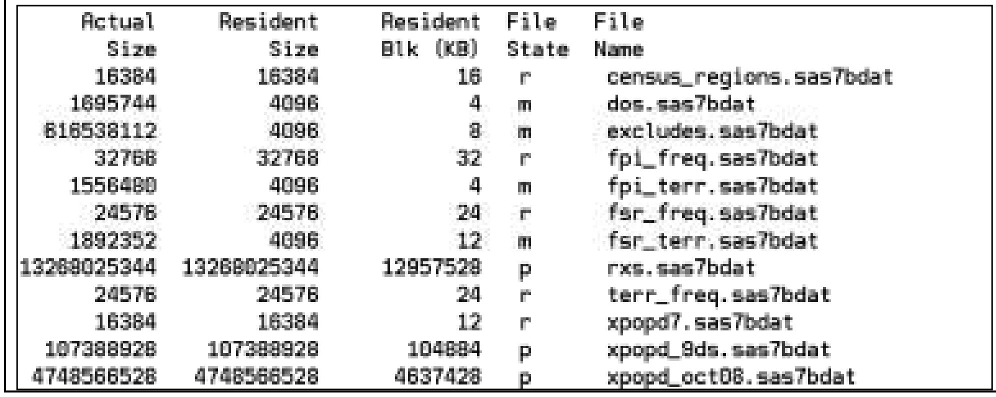
```
IBM Tivoli Storage Manager
Command Line Space Management Client Interface
  Client Version 5, Release 4, Level 2.0
  Client date/time: 03/02/09   19:46:11
(c) Copyright by IBM Corporation and other(s) 1990, 2008. All Rights Reserved.

      Actual      Resident      Resident  File   File
        Size          Size      Blk (KB)  State  Name
 13268025344          4096            12   m       rxs.sas7bdat
```

After issuing a dsmrecall command, the file has been restored to a pre-migrated state:



```
      Actual      Resident      Resident  File   File
        Size          Size      Blk (KB)  State  Name
 13268025344  13268025344      12957528   p       rxs.sas7bdat
```

Results of dsmls command showing all three file states:



```
      Actual      Resident      Resident  File   File
        Size          Size      Blk (KB)  State  Name
       16384         16384            16   r      census_regions.sas7bdat
     1695744          4096             4   m      dos.sas7bdat
   816538112          4096             8   m      excludes.sas7bdat
       32768         32768            32   r      fpi_freq.sas7bdat
     1556480          4096             4   m      fpi_terr.sas7bdat
       24576         24576            24   r      fsr_freq.sas7bdat
     1892352          4096            12   m      fsr_terr.sas7bdat
 13268025344   13268025344      12957528   p      rxs.sas7bdat
       24576         24576            24   r      terr_freq.sas7bdat
       16384         16384            12   r      xpopd7.sas7bdat
   107388928     107388928        104884   p      xpopd_9ds.sas7bdat
  4748566528    4748566528       4637428   p      xpopd_oct08.sas7bdat
```

While the restoration of migrated files usually only takes a few minutes, at most, there have been instances where this delay is significantly longer. When that occurs, it is usually attributed to a hardware problem or a tape that is physically damaged. On one of those rare occasions, the robot that retrieves the tapes actually dropped one and ran it over!!! That was found to be the result of a tape drive that was slightly off level which resulted in tape/tape-gripper alignment issues but that issue has since been fixed with no recurrence of trampled tapes. When there are problems with the tapes, the backup is retrieved by the data center and the file is restored within twenty-four hours, and usually in significantly less time than that.

## TAKING THE NOT-SO-GOOD WITH THE GOOD

As with everything else in life, there are a few drawbacks to TSM, although the benefits (e.g expanded virtual storage space) significantly outweigh the risks. The first risk, as described above, is the occasional damaged tape. Because TSM uses tape storage media and it is being physically mounted and unmounted by a robot, there is the opportunity for things to go wrong and tapes do get damaged. But, again, we have a terrific Unix team who have always been able to restore files from damaged tapes within 24 hours. And we've never had a situation where the file was not retrievable from the backup.

A second "undocumented feature" of our current usage of TSM is that if we are filling up a fileshare extremely quickly, TSM may not be able to migrate enough files fast enough causing the program writing the file to crash as the disk space is used up. TSM has parameters to establish thresholds for when it needs to start archiving files more aggressively so the scenario described above (where disk space gets completely used up), has only happened twice since TSM was put in place.

A third instance occurs when TSM restores a file and then re-migrates it before the programmer has had a chance to process it. This happens most often when reading a file multiple times but not modifying it, because the date/time stamp doesn't change and TSM therefore still thinks it's a candidate for migrating. This situation brings up the concept of the working set of files in play at any given time. Consider an afternoon when 3 different programmers are working on files from within the same filesystem, and they're each switching between 4 different 50 GB files repeatedly. That's 3 programers * 4 files * 50 GB/file, or 600 GB of file in the current "working set". If the files happen to reside within a 500 GB filesystem, at least two of the files from the current "working set" (more likely at least 3) must reside within the tape based portion of the filesystem. There are parameters which can be set at the filesystem level which basically say "hands-off" if files have been modified within a certain period of time. However, setting these features can result in a sort of deadlock situation, where the volume of files newer than the minimum age of files to be migrated exceeds the size of the disk based portion of the filesystem. At that point, TSM is unable to migrate files, the filesystems will reach 100% capacity and programs will fail.

Lastly, while TSM has proven to be a major benefit for giving us a current ratio of almost 50 times more space than the physical disk(s) would allow, it also has "allowed" us to get a bit lazy in cleaning up after ourselves. As a result we probably have a lot more datasets being migrated to tape than are really needed. Despite that, we have a very responsible group of programmers who are fairly diligent (most of the time:) in making sure that we aren't wasting space, be it disk or tape. To that end, here are a few other ways we've been able to effectively manage the large data volumes we need to process.

## USING THE CLEANWORK UTILITY ON WORKSPACE

In addition to being challenged by storage of permanent datasets, work/temp space is another area of concern. We have a 1 terabyte fileshare (non-TSM managed) reserved for workspace and most of the time, that's more than enough. However, there have been several occasions, during monthly processing cycles or when processing an exceptionally large project, when the dedicated fileshare has filled up. Part of the problem is that SAS work and utility libraries can be left behind if a SAS process/program terminates abnormally and they will remain until someone manually deletes them. The SAS cleanwork utility has helped in this regard by automating the cleanup of these remnant work/utility directories. Cleanwork is not perfect and does not always delete everything so it is still necessary to continue to monitor the work fileshare and manually delete work/utility directories that do not get deleted by cleanwork.

Cleanwork needs to be run by a superuser or owner of the work directory but our Unix team has set it up to run nightly in a crontab. Prior to SAS 9.1.3, cleanwork occasionally deleted active work directories but a hot fix resolved the problem and there have been no such experiences since it started running nightly.

Syntax: `cleanwork` *`directory-1`* `<...` *`directory-n`*`>`

## COMPRESSING FILES

Prior to TSM, the mantra in our group was "compress, compress, compress" and, although it has become less critical because of TSM, it is still important. Compression can be accomplished with the Unix compress command or with the SAS compress option. Both commands are easy-to-use and uncompression is equally straightforward. SAS *does not* read files that have been compressed with the Unix compression so only use the Unix compression when the file is no longer needed by any active projects. SAS can read data compressed with its own data compression but while it reduces storage needs for the file and also reduces I/O operations needed to read/write the file, CPU needs are increased for processing the data. There are also occasions where the filesize actually increases when using SAS compress. Bottom line is that compression should be used wisely and monitored carefully, especially in an environment where there are many users vying for CPU/memory resources.

Unix compress syntax: `[un]compress` *`file-1`* `<...` *`file-n`*`>`
SAS compress data set option syntax: `COMPRESS=NO | YES | CHAR | BINARY`

## DEVELOP PROGRAMS USING A SUBSET OF DATA

To speed development and reduce space requirements during the development phase of projects, the LifeLink programming group will typically use a subset of the data for development purposes. The easiest way to do this is using the *OBS=* system or data set option:

```
OBS= n | nK | nM | nG | nT | hexX | MIN | MAX
```

However, when, for example, the data contains multiple observations per patient, you want to make sure you're processing all of the observations for a particular patient and you can't be sure of that if you just randomly select the first 1000 observations. That is, you might only be processing 1 out of 50 observations for a patient because that patient's data starts at observation number 1000. A better approach is to take a random sampling of unique patient ids and then select all of the records for those patients. We've developed a macro that does just that and the code is included below. The code uses the ranuni function to generate a random number which is then used to select unique ids. The unique ids are saved to a dataset which is then used by the programmer to subset all subsequent input data sets. PROC SURVEYSELECT can also be used to accomplish this task.

```
%macro extractuniquepat(dataset,var,percent);
  proc sort data=&dataset. out=unique nodupkey;
    by &var.;
  run;
**create dataset observation number macro;
  proc sql ;
    select count(distinct &var.) into :pat_n from unique;
  quit;
  %put &pat_n.;
**randomly select patid/imsdrnum without replacement;
  data unique;
    set unique;
    random=ranuni(0);
  run;
  proc sort data=unique;
    by random;
  run;
  data sample(keep=&var.);
    set unique;
    K=round(&pat_n./&percent.);
    if _N_ le K;
  run;
  proc sort data=sample out=sasout.sample nodupkey;
    by patid;
  run;
%mend extractuniquepat;
%extractuniquepat(dataset=rxs,var=patid,percent=10);
```

## KEEP ONLY NECESSARY VARIABLES

Just as subsetting observations can help speed development and reduce space requirements, subsetting variables can be equally beneficial. Determining which variables will be required to achieve the end result is not always an easy task and assumes that the final deliverable has been clearly defined and requirements won't change. We all know this is rare in practice so care should be taken when deciding to drop variables and careful planning is necessary to determine which variables will be needed and which won't.

In addition to simply dropping variables, creating formats or codes for lengthy text variables can aid in reducing space requirements. For example, we may be processing drug names that are 20 characters long but since there are usually only a handful of different drugs, creating a format or code for the drug names can save a significant amount of space. In one instance, eliminating 20 characters per observation for 500 million observations resulted in reducing the filesize from over 55 Gb to 42 Gb, saving approximately 13 Gb of space! Multiply that by 10 or 20 jobs processing at the same time and you can see how quickly that adds up, with the potential for saving significant amounts of disk space.

## PROCESSING LOCALLY INSTEAD OF REMOTELY

As mentioned previously, the LifeLink programming group has the luxury of processing on a Unix box and/or HP blade workstations. Using the space saving techniques described above can enable us to offload our development work from the Unix box to the workstations, thereby reducing the demand on the Unix box. The Unix box can then be used for the "heavy-lifting" of processing the full data sets for creation of final deliverables. There are several ways one can move processing from the remote machine to a local machine but we mostly use PROC DOWNLOAD/UPLOAD and/or FTP software such as Ipswitch WS-FTP Pro. Because space is even more limited on a local machine, it's necessary that the subsetting described above occur first, with final processing being done on the Unix box.

## BE A GOOD DATA CUSTODIAN

While many of us have a tendency to keep everything "just in case", the reality is that the majority of the data sets kept during processing are not needed. Most of us don't heed proper data management advice when we're first starting out SAS programming and we end up carrying a rather costly habit with us throughout our programming careers. While it is true that space is relatively cheap, over time this habit can become rather costly. Anyone who has been faced with missing a client deliverable because of space issues can tell you that the cost far outweighs the benefit. Replacing the tendency to keep every data set created during a job “just in case” with a few simple good data custodian habits will make for an efficient and considerate programming environment:

- Only create permanent data sets when absolutely necessary. Carefully consider whether or not anyone will need to reference a data set you’re considering making permanent.
- Consult with other members of the project team to determine which files may be useful for QC and for further analysis and should therefore be saved.
- Delete unneeded data sets promptly, using the delete option in PROC DATASETS.
- SAS 9.2 has a breakpoint feature that allows a program to resume processing in case it should fail. Consider using this as an alternative to saving permanent datasets after each PROC or DATA step.
- Plan out your work – that is, before starting to write a program, create a flow diagram of what you plan to do and map out the datasets you’ll need along the way.

## PLAN YOUR WORK

Did I mention that you should always plan out your work? So I did. Well, let me say it again – and again, and again. One of the best things about the SAS programming language is how easy it is to use and one of the worst things about the SAS programming language is how easy it is to use. Most of us who are now SAS programmers did not set out to be SAS programmers, but have come to be SAS programmers by way of some other career path. Therefore, many of us don’t have a formal education or training in a computer science or software development discipline. Some of the things that one learns in a formal programming or software development curriculum are very valuable and should be applied to SAS programming. One of the most important is writing out the steps to be taken to accomplish the task at hand i.e. a flowchart or pseudocode. No one would build a house without blueprints, or plans, so why would you attempt to write a program to analyze the incidence of liver failure in patients taking a medication for diabetes without doing the same? Is that any less important than building a house? But because SAS is so easy to use and we have not been properly trained in software development practices or the deadlines are so tight or the requirements are so straightforward, etc., etc., we more often than not jump right in and start coding based on nothing but a rather ill-defined set of requirements. Thus the process ends up being less efficient, potentially impacting quality and jeopardizing deadlines. So make it a practice to spend some time to plan out what you’re going to do and try to document beforehand where you’re going to apply the practices noted herein.

## CONCLUSION

While not every programming group is faced with the challenge of processing 100 Gb data sets, taking advantage of the many space management techniques and tools available from SAS and its partners can help any group reduce processing needs and, in turn, reduce costs. And while the techniques outlined in this paper can be used by anyone, what is best for our group may not be the best for yours, so consultation with coworkers as well as external resources is critical to developing well-informed solutions to your specific needs. Additionally, what may be beneficial for one process and one person may not be beneficial to the group or organization as a whole. One thing we’ve learned over time is that impact of the steps you as an individual programmer take impact the group as a whole and that consultation with the other group members is critical. Scheduling, planning and communication are key, with the latter probably being the most important.

Despite the significant progress we’ve made over the past year or so in the LifeLink programming group, we must remain diligent about the way we process these large amounts of data and continue to find new and innovative ways of managing our processing environment. In this challenging economy and ever-changing marketplace, we must continually invest the time and energy to improve our awareness of the tools and techniques available to us. To that end, if you are unsure about what may be best for your organization, SAS and its partners have many capable individuals who would be more than willing to assist you in solving your specific problems. One of the most beneficial actions we took was to bring in a consultant from SAS to evaluate our processing environment and determine what we could do to improve and maximize our processing capabilities. It was an invaluable exercise that has resulted in less downtime and more productive and happy programmers.

## REFERENCES

TSM for Space Management: http://www-01.ibm.com/software/tivoli/products/storage-mgr-space/
Unix crontab info: http://www.adminschoice.com/docs/crontab.htm
HP Blade link: http://www.hp.com/sbso/busproducts_blades.html
HSM Basics; A Presentation by Kevin Low and Marques DeVoe, IBM Software Group, Dec-2007

## ACKNOWLEDGEMENTS

I would like to thank IMS for giving me the opportunity to present at SAS Global Forum 2010 and for providing an environment which is open to investigation and learning. I would especially like to thank all of the capable staff in the Unix Admin group: Dave, Linda, Michael, Donna and I’m sure there are others that I’m missing.

Of course, I have to thank all of the programmers in my group: Usha, Julia, Lee, Bernie, Stu, Nick and our adopted son, Solomon. And, of course, my boss Scott for putting up with all of us!
There are so many others who have made this possible so thank you all – you know who you are.

## CONTACT INFORMATION

Your questions and suggestions are valued and encouraged. Contact the author at:

Paul Doucette
IMS
960A Harvest Drive
Blue Bell, PA 19422
voice: 610.832.5827 fax: 610.832.5850
<pdoucette@us.imshealth.com>
http://www.imshealth.com