

IAM

C O N C E P T S & F A C I L I T I E S G U I D E

t a b l e o f c o n t e n t s

Introduction	3
PART ONE. An Overview of VSAM.....	5
Introduction.....	5
Defining a VSAM File	5
The VSAM File Structure	6
Loading the Data	6
CI% FREESPACE	7
Adding Records to the File.....	7
CA% FREESPACE	7
Inserting 'HIGH KEY' Records.....	9
DASD Utilization	9
Using Buffers	10
PART TWO. VSAM Problems.....	11
Introduction	11
CI/CA Splitting.....	11
DASD Usage	12
BUFFER Usage.....	13
Summary	14

t a b l e o f c o n t e n t s

PART THREE. IAM	15
Introduction.....	15
IAM: The Power Over VSAM.....	15
The VSAM Interface (VIF).....	16
Creating an IAM File: IDCAMS Support.....	16
IAM Overrides.....	17
The IAM File Structure	18
File Reorganization.....	19
Using the Optional 'Data Compression' Feature	20
Support for z/OS Hardware Compression.....	20
Space Release	20
Support for PAV	20
'In-Core Index', 'Real Time Tuning' and 'Dynamic Tabling'	21
IAM Under CICS.....	22
IAM Execution Time Statistics	22
Special Feature: Record Level Sharing (RLS).....	23
Special Feature: Alternate Index (AIX) Support	24
ESDS Support	25
Identifying Candidates for IAM Conversion.....	26
SMS Support	27
IAM Reliability.....	27
IAM Installation.....	28
IAM ISPF Panels	28
Summary	28
Appendix	29

INTRODUCTION

CONCEPTS & FACILITIES GUIDES

For more than 30 years, Innovation Data Processing has been producing high-quality Storage Management Software. Over the years, its products have evolved into today's ultra high-speed, safe, reliable storage management solutions for OS/390, z/OS, LAN and Open Systems Data.

It all started with the **FDR Storage Management Family**, of which over 5000 licenses have now been sold worldwide. The FDR Family is the complete Storage Management System for OS/390 and z/OS.

FDR has become the industry standard for fast, reliable backups of MVS OS/390 data.

ABR adds a layer of automation to the standard functions of FDR, providing advanced backup facilities like *Incremental Backup*, *Application Backup* and *Archiving*.

COMPAKTOR and **FDRREORG** further enhance the suite by adding intelligent and powerful reorganization processes, for whole DASD volumes and for Sequential, PDS and VSAM datasets.

FDREPORT provides extensive customized DASD Management Reporting to suit many needs and purposes.

FDRCLONE is an extension to ABR, providing the ability to "clone" volumes and/or datasets on a test or disaster recovery system. It includes **FDRDRP**, a utility that can reduce ABR full-volume recovery time by up to 80%.

FDRINSTANT enables FDR/ABR to take *non-disruptive backups* of offline volumes, created by the latest DASD Subsystem features like StorageTek/IBM SnapShot Copy, EMC² TimeFinder/BCV, HDS ShadowImage and IBM FlashCopy.

FDRPAS (FDR Plug and Swap) allows for the non-disruptive movement of OS/390 disk volumes from one disk device to another. When new disk subsystems are installed, active online disk volumes can be swapped to drives in the new subsystem without disrupting normal operations or requiring a re-IPL. This allows a 24 x 7 installation, with no window for major re-configurations and hardware changes, to install and activate new hardware.

THE FDR/UPSTREAM Family of Products builds on the strengths of the FDR Storage Management Family providing a fast, safe and reliable solution to backing up Open Systems data from file servers and workstations, across a network connection to disk or tape on the OS/390 host. If the Open Systems data is resident on an EMC² Symmetrix with Enterprise Storage Platform (ESP), **FDRSOS** and **FDR/UPSTREAM/SOS** products provide additional performance enhancements to the backup and restore process by utilizing high-speed mainframe channels.

IAM is Innovation's alternative to VSAM KSDS, ESDS and (as a cost option) AIX files. It eliminates VSAM performance bottlenecks and reduces VSAM file sizes by more than 50%.

FATS/FATAR and **FATSCOPY** are a set of multi-purpose tape subsystem Media Integrity tools that allow for online tape certification, verification and erasure, as well as the ability to analyze and copy tapes.

INTRODUCTION

CONCEPTS & FACILITIES GUIDES

Each of the Innovation products are described in a range of Concepts & Facilities Guides that have been created by the Innovation UK office, but which are available *free of charge* from your local office (see back cover for details).

In this particular guide, we look at **IAM**.

PART ONE and **PART TWO** give a basic overview of the fundamentals of the VSAM file structure and explain how this structure is the cause of most of VSAM's performance problems.

Note: These first two sections are aimed at readers who are unfamiliar with VSAM and who would like to gain a better understanding of its design and structure.

Readers who are already familiar with VSAM may want to go directly to PART THREE.

In **PART THREE** of this guide we look at how IAM can be used to provide a quick and relatively cheap solution to the problems caused by VSAM.

*Any comments or suggestions regarding this guide can be directed to:
support@fdriinnovation.com*

PART ONE

AN OVERVIEW OF VSAM

*Note: Readers who are already familiar with VSAM may wish to skip **PART ONE** and **PART TWO** and go directly to **PART THREE**.*

Introduction

IBM's primary access method, VSAM, has been around now for several decades, yet it is still frequently used as the basis for high-profile 'bought in' packages—such as Financial and Personnel applications—as well as numerous 'home-grown' systems. The very highest levels of a company's Management (e.g. Financial Directors) can often use these applications, as well as the regular workforce.

The design of VSAM directly affects the performance and efficiency of these systems. Common problems caused by VSAM are an increase in Response Times during the online day and/or an unacceptably long overnight Batch Window, which then impacts the availability of the online system the following morning. The use of DASD space can also be an issue, especially if users of an application are charged for their DASD consumption. Some files can grow very large—to several Gigabytes and more.

When these problems are being experienced, the users of the system (including Management) are unlikely to accept, or even understand, that the cause is due to the VSAM file structure. Equally, they will find it unacceptable to be told that, because of the nature of VSAM, there is little that can be done without a great deal of manual intervention and tuning. They will want quick solutions. Unfortunately, traditional 'quick' solutions like hardware upgrades, can only give short-term relief and are often very expensive to implement.

In order to understand fully the problems inherent in VSAM, it is first necessary to take a look at the VSAM file structure, as it is here that the root cause of the problems exists. We will concentrate on the structure of VSAM KSDS's although, as you will see in PART THREE, IAM also supports Alternate Index (AIX) and ESDS datasets. Readers already familiar with VSAM may wish to bypass the rest of PART ONE and move on to PART TWO on page 11. It is assumed that *all* readers have a basic understanding of Catalogs, VVDS's and VTOCs.

Defining a VSAM File

Let's start by looking at how VSAM files are created. Under DFSMS, this can be done through regular JCL but, for the purposes of this overview, we will assume that IDCAMS is still the preferred method. The following JCL extract illustrates a DEFINE of a single Index VSAM KSDS using IDCAMS:

```
//DEFVSAM EXEC PGM=IDCAMS
//SYSPRINT DD SYSOUT=*
//SYSIN DD *
DEFINE CLUSTER
  (NAME(MY.TEST.DATASET) -
  VOLUMES(MVS001) -
  SPEED REUSE ) -
  DATA (RECORDS(5000 500) -
  FREESPACE(10 20) -
  KEYS(10 0) -
  RECORDSI ZE(256 256) -
  CISI ZE(4096))
/*
```

PART ONE

AN OVERVIEW OF VSAM

The IDCAMS job on the previous page would result in a VSAM KSDS cluster being created with the name 'MY.TEST.DATASET'. This cluster would be comprised of two components:

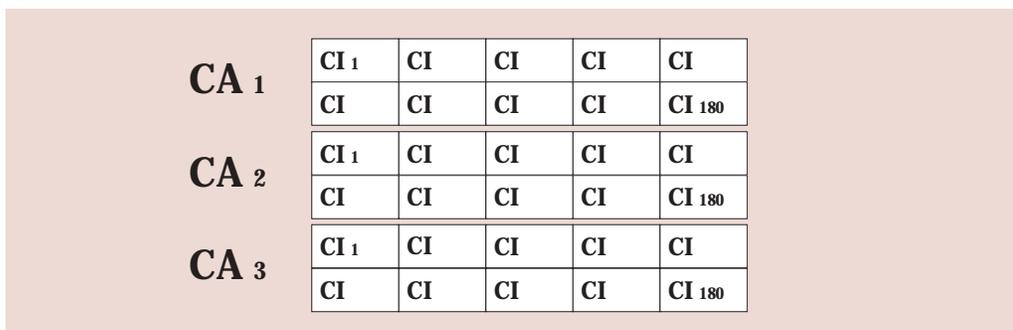
- The '**DATA**' Component: Containing the data records for the file, stored in *key sequence*, hence the name Key Sequence DataSet (KSDS).
- The '**INDEX**' Component: Containing records indexing the data component.

In our example job both components would be allocated on MVS001, the cluster would be cataloged to MVS001 and an entry placed in the VVDS on the volume for both the Cluster and the individual Component names. In addition, DSCB's for the component names would be placed in the VTOC on MVS001. It is not, however, this *recording mechanism* for VSAM which is the cause of its poor performance. Instead, it is the internal structure of the file itself.

The VSAM File Structure

Within the DATA and INDEX components, an internal structure is created at the time of the DEFINE. We will be concentrating on the DATA component, but a similar design (and resulting performance problems) can be experienced in the INDEX component as well.

The DATA component is divided into Control Areas (CA's) that are, in turn, divided into Control Intervals (CI's). A typical CA size is 1 CYLINDER and the CI's contained within each CA are of a size defined by the CISIZE parameter coded in the DEFINE. In our example earlier, if the file was allocated on a 3390, twelve 4096 CI's would fit onto a track and, as the 3390 has 15 tracks per cylinder, the CA would contain 180 CI's. The internal structure of the DATA component would therefore look something like this:



Loading the Data

Once the file has been DEFINE'd, the data records must be loaded into it. Remember that the data is stored in a KSDS in *key sequence* order. The initial loading of data is usually done using an IDCAMS REPRO, as shown below. In the following example, the input data is coming from a previously created sequential file (SEQFILE) held on disk, and is going out to a VSAM file called 'NEW.VSAM.FILE' which has already been DEFINE'd in a previous step or job.

```
//DEFVSAM      EXEC  PGM=IDCAMS
//SYSPRI NT    DD   SYSOUT=*
//SEQFILE     DD   DSN=seq. file, DISP=SHR
//SYSIN       DD   *
              REPRO INFILE(SEQFILE) OUTDATASET(NEW.VSAM.FILE)
/*
```

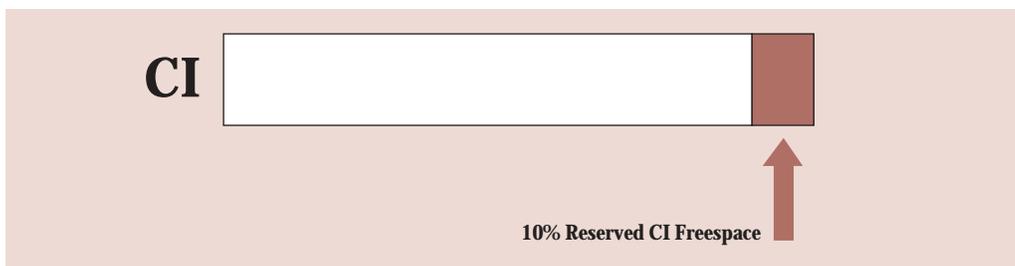
PART ONE

AN OVERVIEW OF VSAM

CI% FREESPACE

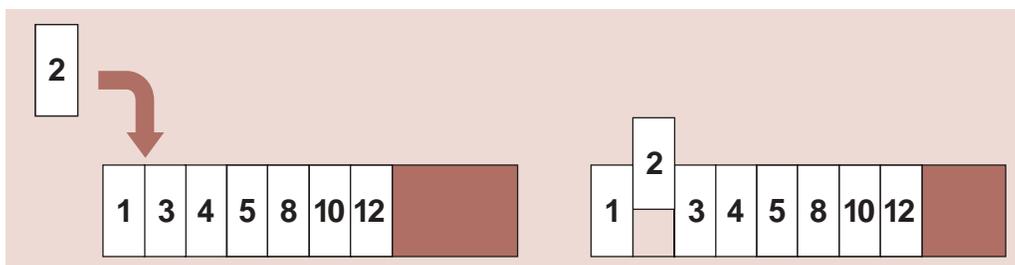
When the KSDS is DEFINE'd, a decision has to be made as to whether any freespace must be reserved in the file during the initial LOAD. If the KSDS will only be used for READ purposes, there would not be any need to reserve freespace. However, if records are to be inserted into the file (or increased in size as a result of being updated), space must be reserved to allow these inserts or updates to take place.

The first type of freespace available in a VSAM KSDS is called CI% FREESPACE. This is reserved when the file is DEFINE'd, via the FREESPACE parameter. In our earlier IDCAMS example, we requested CI% Freespace of 10%. When REPRO loads each CI with data, 10% of the space would be reserved for future inserts. After the load has completed, every CI in every CA throughout the whole KSDS would look something like this:



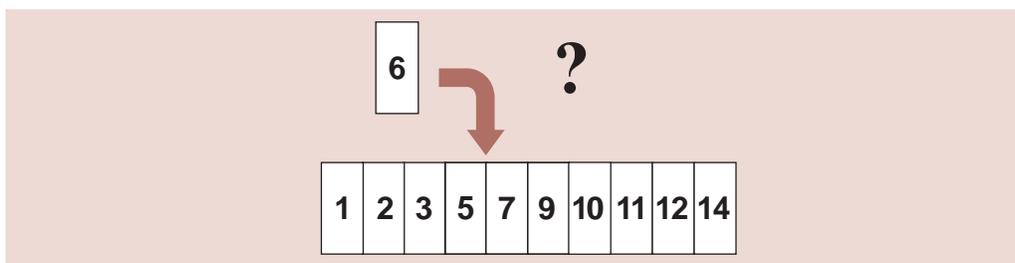
Adding Records to the File

As additional records are subsequently added to the KSDS *after* the initial load, VSAM will attempt to place them into the appropriate CI. In the example below, a record with the key of '2' is to be added to the KSDS. To maintain the key sequence, it must be placed in the CI, which already contains records with keys of '1' and '3'.



CA% FREESPACE

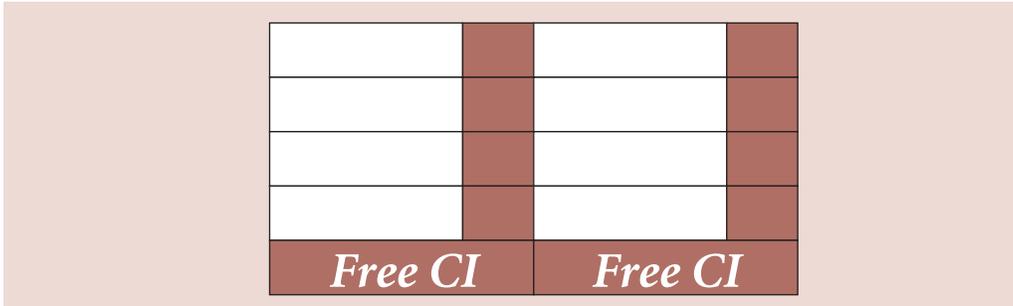
Depending on the level of insert activity, and the amount of freespace coded on the DEFINE, it may be possible to exhaust the available freespace in a CI, such that additional inserted records cannot fit into the correct CI.



PART ONE

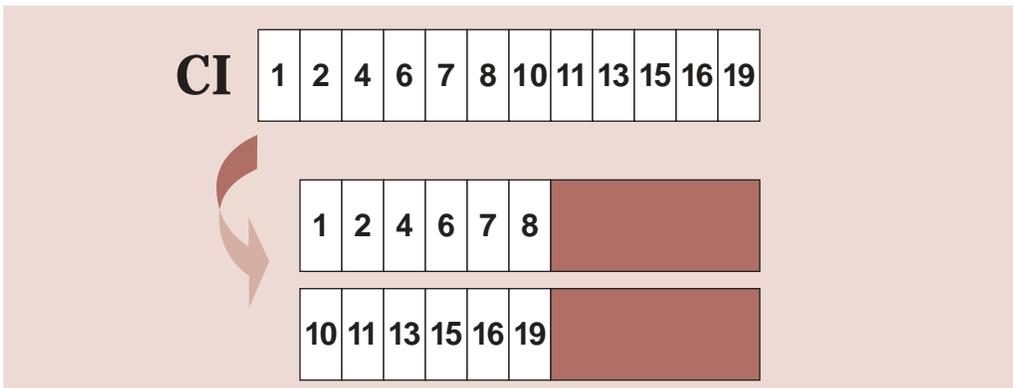
AN OVERVIEW OF VSAM

To cope with this, another type of freespace, called CA% FREESPACE, is also reserved in the file. This is achieved using the second value on the FREESPACE parameter in the IDCAMS DEFINE and it involves reserving *whole free CI's* within a CA during the load process:

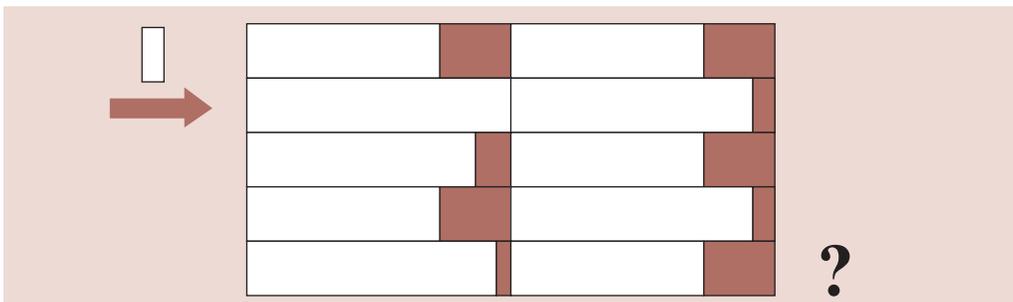


When a CI has exhausted its CI% Freespace, it is divided into two in an operation known as a 'CI Split'.

Approximately half of the records from the full CI are moved into one of the reserved free CI's at the end of the same CA (see *below*). Once the CI Split has completed, the new record that caused the split to occur can then be inserted into the original CI.



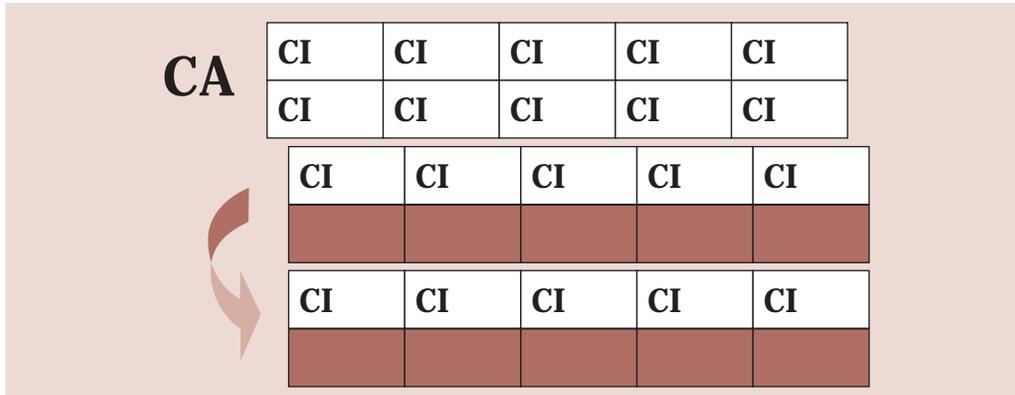
Unfortunately, CA% freespace can also become exhausted. If, for example, a large number of records are being added to a file, and they all have a similar key value, it is quite common for *all* of the free CI's in a CA to become used as a result of the CI Splitting (see *below*). How then can additional records be added?



PART ONE

AN OVERVIEW OF VSAM

Once all free CI's in a CA have been exhausted, new CI's *cannot* be used from another CA. Instead, VSAM has to do what is known as a 'CA split':



In the 'CA split' process, a new CA is formatted at the end of the file to include the same CI and CA freespace as used throughout the rest of the file. Once the new CA has been formatted, half of the CI's in the original CA are moved into the new one (*see above*).

Unfortunately, it doesn't end there. Once the CA split has completed, a CI split still has to occur in the *original* CA to allow the original CI to take the inserted record! Only when the CI split has completed successfully can the record finally be added to the file.

PART TWO of this Concepts Guide highlights the effects that this CI/CA splitting can have on the performance of the KSDS, both during the split and also when the records are later being processed. It also discusses the inadequacy of the two commonly used alternatives for reducing the number/effect of splits—namely '*Increasing freespace*' and '*Running regular REORGs*'. PART THREE of this Guide then shows how the IAM file structure handles inserted records with a much greater level of efficiency.

Inserting 'HIGH KEY' Records

So far, we have concentrated on inserting records into the file which have keys *lower* than the 'last' record in the file when it was first loaded. If records with *higher* keys are to be added to the end of the file, they are written to the last CA. When that CA becomes full, new CA's will be formatted. Eventually, assuming it was allowed within the DEFINE, the VSAM file would start to take additional secondary extents (up to 123 per volume) to cater for new 'high-key' records being added to the file.

DASD Utilization

In the Introduction, we suggested that VSAM's inefficient DASD utilization can be as much of a problem to users as the performance issues highlighted above. Indeed, the two problems can often go hand-in-hand. For example, PART TWO of this Guide shows how increasing the freespace values for the file can reduce the CI/CA splitting described above, but can then lead to excessive DASD usage/wastage.

PART ONE

AN OVERVIEW OF VSAM

In addition, VSAM also wastes DASD space in the following situations:

- When the file contains records with **Large Keys**. VSAM can waste CI's in every CA when handling these.
- When **inefficient blocksizes** are chosen. The blocksize of a VSAM KSDS must be a multiple of 512 or 2048 bytes. This means that KSDS's are often created with blocksizes that are not the most efficient for the DASD device on which they reside.
- When a **small Index CISIZE** is used in conjunction with a **small Data CISIZE**. In the Index component, a CI *must* index all of the CI's within an individual CA in the Data component. If the Data CISIZE is small (*i.e.* lots of CI's in the CA) the Index CI referencing it can fill up, leaving the remaining CI's in the Data CA unused.

Later, in PART THREE of this Guide, we show how IAM's efficient file structure, default blocking factors, and optional Data Compression and Space Release features can be used to utilize DASD more effectively.

Using Buffers

When the data in a VSAM file is to be processed, either by an online system (e.g. CICS) or by a batch job, the data records are moved into areas of virtual storage called 'buffers'. Holding records in these buffers eliminates the need for I/O operations (to disk) to access the data records in the file.

The number of buffers that can be used to hold INDEX and DATA records can be controlled by the user, either through the BUFSP parameter on the IDCAMS DEFINE, through the AMP parameter in JCL, or hard-coded within the actual accessing program itself. Under CICS, a 'pool' of buffers can be created and shared by various VSAM files.

Although it may be generally true that the more buffers made available, the better the performance of a VSAM file will be, this is not always the case. Indeed, it is important to take into account the *type* of access being done against the file—sequential or random. This will often vary between CICS during the day and batch at night. It is not, therefore sufficient to give *all* VSAM files *maximum* buffers; this would have a serious impact on the overall consumption of storage. More care has to be taken to ensure that the files that need the buffers get them, and the ones that do not, will not.

Unfortunately, some of the methods described above for specifying the number of buffers (BUFSP, AMP, etc) are manual processes. PART TWO of this Guide highlights the problems caused by this and PART THREE shows how IAM's '*In-Core Index*', '*Real Time Tuning*' and '*Dynamic Table*' mechanisms can alleviate these problems.

That completes our brief overview of VSAM. In PART TWO, we move on to highlight the problems inherent within the VSAM file structure and explain the difficulties facing those responsible for the management, control and tuning of VSAM files within an application.

PART TWO

VSAM PROBLEMS

Introduction

PART ONE of this Guide gave a brief overview of VSAM and showed how the internal file structure is the root cause of the most common VSAM problems—namely *Performance* and *DASD Utilization*. Here, in PART TWO, we take a closer look at those problems and assess the implications they have on the Applications that use VSAM.

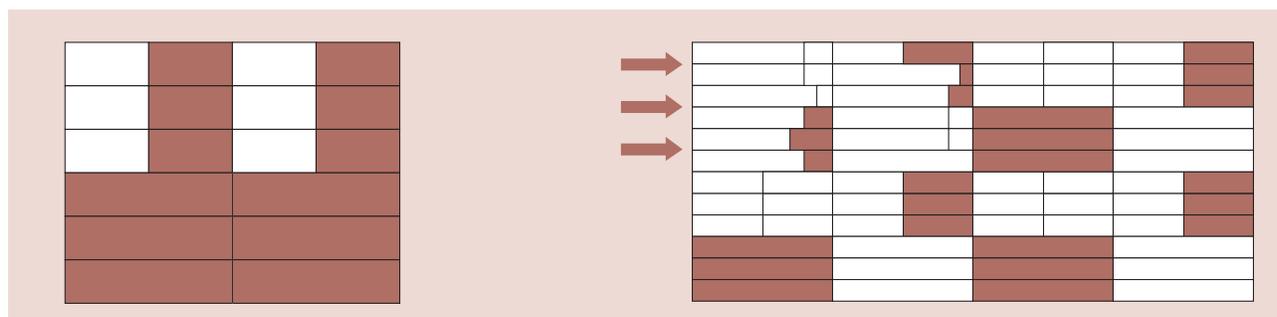
CI/CA Splitting

PART ONE described the concept and resulting inefficiency of VSAM's CI/CA splitting mechanisms. This inefficiency can be described as:

- An overhead on CPU usage and I/O's, both to complete the split and to process the records that have been placed in a split CI/CA.
- An increase in the Response Times of online systems and Elapsed Times of batch jobs as a result of the delays during the split, and subsequent processing of a split file.
- A potential exposure to data integrity. VSAM clusters can become unusable if an insert is interrupted during a CI or CA split because records in the CI or CA being split may be lost if the split does not complete successfully.

One method used to counteract CI/CA splitting is to increase the freespace definitions in VSAM files that are to take a high number of inserts, with a view to decreasing the number of CI/CA splits.

For example, 'FREESPACE(50 50)' could be coded to request that 50% of *each* CI be left free and 50% of *all* CI's in *every* CA be left empty (see below, left). This would, however, only reduce the number of CI/CA splits that occur if the new records are being inserted evenly throughout the file. This is rarely the case! Often (see below, right), the insertion of new records is concentrated around a specific range of keys. Freespace cannot be graduated around the file, so if the inserts are clustered, splitting will *still* occur around the area of insert activity, while other areas of freespace go completely unused.



An alternative for reducing the effect of CI/CA splitting is to reorganize the file on a regular basis. *Reorganization* is a process where IDCAMS reads all of the data records from the KSDS containing CI/CA splits and reloads them in proper key sequence, replenishing the freespace areas as it goes. Depending on the size of the KSDS, a Reorganization can take a considerable amount of time (*i.e.* tens of minutes) and while it is taking place, no other users or applications can access the file.

PART TWO

VSAM PROBLEMS

As we pointed out in the Introduction to PART ONE, VSAM files are often used in very 'high-profile' systems, for which such prolonged unavailability is undesirable and, in extreme cases, can result in a loss of company business. The need to REORGANIZE therefore creates two major headaches for the personnel responsible for the performance and availability of VSAM in the systems concerned:

- **REORG vs PERFORMANCE**

On the one hand, they can keep the file available 'non-stop' for several days, at the expense of letting it split and suffer ever-decreasing performance. Alternatively, they can opt to REORGANIZE the file(s) each night, potentially causing a delay in the availability of data and, in addition, adding work to an overnight schedule that may already be stretched to the limit.

- **SELECTING FILES FOR REORG**

The *selection* of files for REORG is a manual task with IDCAMS and is open to error. Some files that really need reorganization may, for example, get missed (and continue to perform badly), while others that don't need reorganizing still get REORG'd (causing unnecessary downtime and adding to the total REORG time for the system).

The FDRREORG component of the FDR DASD Management System can be used to help with these problems. First, it automates the selection of files requiring reorganization (based on user criteria) to ensure that all the files which need REORGing will be picked up, while those that don't can be bypassed. This ensures their continued availability and reduces the total REORG time. Secondly, FDRREORG can then do the actual reorganization of the selected files much faster than IDCAMS. For more information on the functions and benefits of FDRREORG, see the '*FDR Storage Management Family*' Concepts and Facilities Guide.

PART THREE of *this* Concepts Guide shows how IAM can also be used to further reduce the effect of these problems by handling inserted records in a much more efficient manner, thus reducing the processing overhead when REORG's are run.

DASD Usage

As highlighted in PART ONE (and again earlier in this section), VSAM is not the most efficient user of DASD Space. Its freespace mechanism can waste large areas of space, unless records are inserted evenly throughout the file. In addition, VSAM has a limited ability for specifying the blocksize of files for optimum DASD usage.

DASD space may be cheaper '*by the GB*' these days, but few companies can afford to unnecessarily waste it. And, of course, it is never just a case of the purchase price of the DASD. Consideration also has to be made on the increased hardware maintenance costs, the increase in backup times for the 'over-large' files and, perhaps more importantly, the increase in restore and disaster recovery time.

With an increasing demand for keeping more historical data online, the main files in some application systems can easily grow to several Gigabytes in size. When VSAM files reach this size, performance problems and poor DASD utilization becomes all the more apparent.

PART THREE of this Guide describes how IAM can be used to:

- Store data in significantly less space than required for VSAM
- Improve the performance on larger files

PART TWO

VSAM PROBLEMS

There are two other 'DASD Usage' issues that are common problems for VSAM:

- Over-allocation: It is difficult to calculate the space required for a VSAM file. Users often over-allocate, or use a standard IDCAMS DEFINE job to create their VSAM files, without first altering the space allocation to match each individual file's requirement.
- Additional extents: VSAM files can grow to 123 extents. Multi-extent datasets are inefficient and can degrade the performance and utilization of the DASD on which they reside.

These issues cannot be ignored. It is the aim of all DASD Managers and Storage Administrators to ensure that their DASD Space is being used efficiently and they will carefully monitor the thresholds of their DASD volumes. However, because of a lack of suitable tools, many do not look at the actual usage of the *allocated* space within the files themselves. The wasted space inside VSAM files caused by over allocation and the CI/CA freespace mechanism can often go unnoticed. It is almost impossible, for example, to find out how much of the CA freespace in a file has been used. Even if the DASD wastage and multi-extent problems are recognized and fully understood, the 'cure' is usually a reduction in the freespace definitions, at the expense of more CI/CA splits and the resulting performance degradations described earlier.

These problems can, in part, be addressed using the COMPAKTOR component of the FDR DASD Management System, as described in the '*FDR Storage Management Family*' Concepts And Facilities Guide. Compaktor can be used to release unused over-allocated space in VSAM files and merge the extents of multi-extent VSAM files. PART THREE of *this* Concepts Guide shows how IAM can also be used as an additional solution, by controlling unnecessary over-allocation through its 'Automatic Release' feature.

Buffer Usage

As highlighted in PART ONE, the methods available for specifying the number of DATA/INDEX buffers for a VSAM file are entirely manual. Using these methods, two options are available to decide *how many* buffers to allocate:

- Someone can look at the activity of each individual file and try to assess its buffer requirements. This can cause problems if a file is accessed by several programs (some doing Random I/O, others doing Sequential) which may require different buffering. It also assumes a regular 'check-up' will be made to ensure that the buffer requirements set previously are still valid, particularly if the use of the file has altered in some way.
- Alternatively, a 'rule-of-thumb' mechanism can be used, where buffering is estimated on a past history basis. This may lead to some files getting insufficient buffers, while others are given far more than they need. Neither is satisfactory. The ones needing more buffers will not perform as well as they could and the files with too many buffers will be using resources (*i.e.* storage) that could be better utilized elsewhere.

A further complication arises when there are insufficient skills within a site to carry out the 'manual' buffer tuning described above. Furthermore, as many sites now have hundreds, if not thousands of VSAM files, manually tuning them could easily become a full-time job!

Mechanisms such as DFSMS VSAM System Managed Buffering (implemented via the DATACLAS) and the LSR pooling functions under CICS can help to alleviate some of these problems. However, the creation and specification of these systems is still a manual task often requiring some degree of 'VSAM Skill' to implement and monitor.

In PART THREE we show how the '*In-Core Index*', '*Real Time Tuning*' and '*Dynamic Table*' mechanisms of IAM can remove the requirement for manual buffer tuning and, of course, the exposure to getting it wrong.

PART TWO

VSAM PROBLEMS

Summary

Although some of the problems highlighted in the first two Parts of this Guide can, in part, be addressed by time-consuming manual effort, this is rarely feasible in today's Data Centers where manpower is already stretched. Moreover, many of the 'cures' are simply a *shifting* of the problem from one area to another. Increasing freespace can, for example, reduce CI/CA splitting, but as we have shown, this 'cure' comes at the expense of an increase in DASD consumption and wastage.

It would appear to be a no-win situation for the personnel responsible for VSAM in a site. This is not the case! PART THREE of this Guide describes how IAM provides an alternative to VSAM, to address the problems we've highlighted above.

PART THREE

IAM

Introduction

PARTS ONE and TWO of this Guide highlighted the problems inherent in VSAM. They illustrated how the VSAM file structure can be stressed when used in the high-access, high-profile application systems of today. As such, the problems that were perhaps just minor headaches a few years ago have now become so acute that the manual ‘cures’ which have served well in the past are no longer sufficient. In many Data Centers there are now simply too many VSAM files to handle and too little time to spend manually tuning them.

Several products exist in the market which try to alleviate this problem by removing some of the manual tuning effort. All of these products, however, ignore the fact that the fundamental problem of VSAM is the file structure. The only way to truly address the problems caused by VSAM is to move the data into an alternate, more efficient file structure.

IAM: The Power Over VSAM

The Innovation Access Method (IAM) is a reliable, high-performance disk File Manager, designed as a transparent alternative to VSAM. It will significantly outperform VSAM under z/OS and OS/390 systems and it does *not* require any permanent modifications to the Operating System. It is designed to *co-exist* with VSAM without affecting the normal use of VSAM, and without interfering with VSAM Catalog Management.

IAM supports the concept of **Record Level Sharing (RLS)** within a *single* MVS image, allowing files to be shared (for read *or* update) between multiple CICS regions, TSO users and batch jobs—see “*Special Feature—RLS Support*” on page 23.

IAM offers the following benefits over VSAM **KSDS** files *without* the need for manual tuning:

- 50-80% reduction in I/O's (EXCP's)
- 20-40% reduction in CPU usage
- 30-70% reduction in DASD space usage
- 50-80% reduction in Batch Elapsed Times
- 50-80% reduction in Online Response Times

The “IAM/AIX” cost option provides special support for **VSAM Alternate Index (AIX)** files (see page 24).

From 3rd Quarter 2003, the “IAM/AIX” option also provides support for VSAM **RRDS** datasets.

ESDS files are also supported (see page 25).

In a moment, we will take a closer look at IAM to see how the above benefits are possible. First, we are going to see how IAM files can be used in place of VSAM without requiring any permanent changes to JCL, or to application programs, CICS, or even the Operating System!

The secret is the IAM/VSAM Interface: the VIF...

User Experiences

“IAM reduced VSAM space requirements from 106,000 tracks to less than 53,000 tracks...”

“We converted our VSAM files to IAM without change. Now the file is over 20GB...”

“We replaced our existing VSAM compression package with IAM and saw a 50% saving in CPU time...”

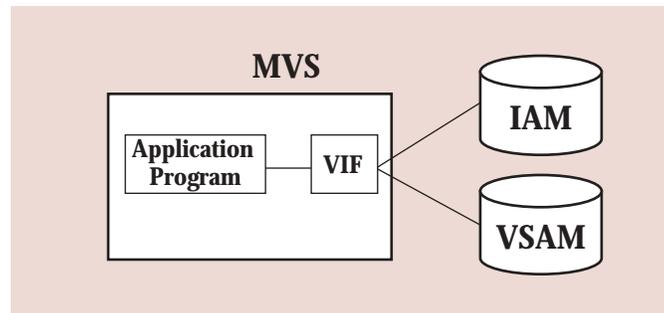
See the Appendix for a list of vendor products that have successfully converted their VSAM files to use IAM.

PART THREE

IAM

The VSAM Interface (VIF)

The VIF allows any program (e.g. Cobol, Assembler, PL/1, RPG) which uses keyed index access against a VSAM KSDS to load and then sequentially or randomly access IAM files without program changes. It is started at MVS IPL time, and dynamically installs itself in the SVC table. It then intercepts the SVC 26 (Locate), VSAM OPEN, VSAM CLOSE (TYPE=T), and SHOWCATs to determine if the request is for a VSAM file, or an IAM file:



If the request is for VSAM, the normal IBM routines are given control and the file processing continues as normal. If the file is IAM, the OPEN of its ACB is simulated to look like a VSAM OPEN. Then, when an online system or batch application program issues a record request against the file (e.g. GET, PUT, POINT, etc) the request will be processed by the VIF to look like the VSAM requests.

Creating an IAM File: IDCAMS Support

The VIF is also designed to intercept IDCAMS DEFINE's if they are 'tagged' as being DEFINE's for IAM files. This allows IAM files to be created using regular IDCAMS DEFINE jobs, despite having an entirely different file structure. (Note: The other methods for allocating VSAM files—JCL, SMS, etc—can also be used to create IAM files. However, as in PART ONE, we will assume that IDCAMS will be the preferred method). The REPRO, DELETE, PRINT and LISTCAT functions of IDCAMS are also supported for IAM files.

Because IAM files can be DEFINE'd with IDCAMS, this allows for a very easy migration from VSAM to IAM. Indeed, as VSAM files are usually DELETE/DEFINE'd on a regular basis during the Reorganization process, the IDCAMS DEFINE jobs are already available. It is a very simple task, therefore, to create IAM files with those jobs.

As you will see later, some of the parameters of the IDCAMS DEFINE are not relevant to an IAM file, but these parameters do NOT have to be removed and are ignored when the IAM file is created. Other parameters (e.g. FREESPACE) may be altered by IAM to take into account its more efficient handling of data. This will also be explained later in this section.

Using IDCAMS, there are 2 methods (shown below) for 'tagging' a DEFINE to create an IAM file. (Note: In the UK, the tag is £IAM):

- Placing '\$IAM' in the OWNER field
- Placing the characters '\$IAM' anywhere within the file name

Either of the above methods can be used each time an IAM file is created. The 'owner' method is the most transparent because the actual filename does not change and, hence, the JCL of jobs that processes it will run without requiring any alteration.

PART THREE

IAM

The 'filename' method can be chosen instead, if there is a particular need to identify files as IAM by their filename. IAM files do not have separate '.DATA' and '.INDEX' components to make them easily identifiable—the index portion of the IAM file is contained within the main file allocation itself. Regardless of the method chosen, the VIF will intercept the DEFINE and an IAM file (with an MVS DSORG=PS) will be created.

In the examples below, the result will be a non-VSAM dataset allocated on MVS001 with the name of 'MY.TEST.DATASET' or 'MY.TEST.\$IAM.DATASET'. This is an example of creating a single index IAM KSDS. See later for details on creating IAM ESDS and Alternate INDEX (AIX) files.

The 'OWNER FIELD' Method

```
//DEFVSAM      EXEC PGM=I DCAMS
//SYSPRI NT    DD  SYSOUT=*
//SYSI N      DD  *
DEFINE CLUSTER
  (NAME(MY. TEST. DATASET) -
   VOLUMES(MVS001)
   OWNER($IAM) -
   SPEED REUSE ) -
  DATA (RECORDS(5000 500) -
  FREESPACE(10 20) -
  KEYS(10 0) -
  RECORDSI ZE(256 256) -
  CI SI ZE(4096))
/*
```

The 'FILENAME' Method

```
//DEFVSAM      EXEC PGM=I DCAMS
//SYSPRI NT    DD  SYSOUT=*
//SYSI N      DD  *
DEFINE CLUSTER
  (NAME(MY. TEST. $IAM. DATASET) -
   VOLUMES(MVS001) -
   SPEED REUSE ) -
  DATA (RECORDS(5000 500) -
  FREESPACE(10 20) -
  KEYS(10 0) -
  RECORDSI ZE(256 256) -
  CI SI ZE(4096))
/*
```

(Note: If required, IAM datasets can be defined as DFSMS extended format sequential datasets, allowing them to fully utilize 3390-9 and other "large" devices, such as the 3390-27. This effectively raises the theoretical maximum size of an IAM file to almost 3 terabytes of compressed data).

IAM Overrides

Some of the optional features of IAM (e.g. Data Compression—see later) are requested at file creation time. This is done by coding an IAMOVRID DD in the IDCAMS DEFINE JCL. The example below shows how to request Data Compression on all files being DEFINE'd in the particular step.

```
//DEFVSAM      EXEC PGM=I DCAMS
//SYSPRI NT    DD  SYSOUT=*
//SYSI N      DD  *
DEFINE CLUSTER
  (NAME(MY. TEST. $I AM. DATASET) -
   VOL(MVS001) -
   KEYS(9 0) -
   RECORDSI ZE(1024 1024) -
   CYLI NDERS(500 200)
  )
/*
//IAMOVRID    DD *
//CREATE      DD=&ALLDD, DATA COMPRESS=YES
/*
```

There is also an 'ACCESS' Override statement available to request further IAM features at file access time (e.g. Dynamic Tabling). These features will be discussed later in this section.

Let's take a look now at the internal structure of an IAM file...

PART THREE

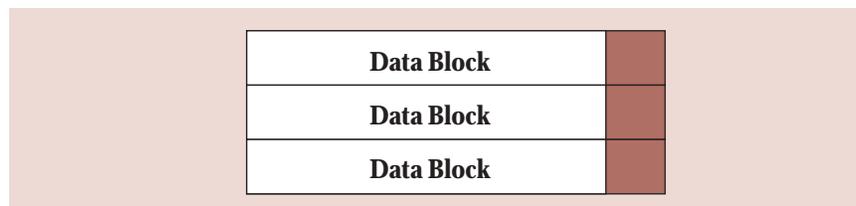
IAM

The IAM File Structure

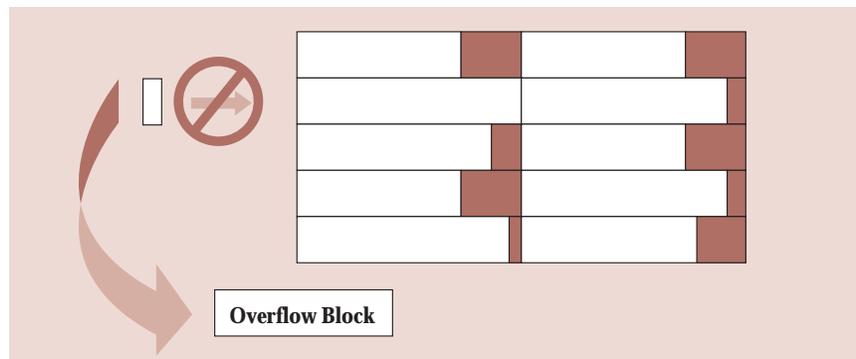
PART ONE of this Guide described the internal structure of a VSAM file and illustrated its inefficiency when handling randomly inserted records. The IAM file structure, on the other hand, is designed to take inserted records more efficiently.

When allocated, an IAM file is divided into logical blocks of a size determined by the CISIZE parameter in the IDCAMS DEFINE, or via the BLKSIZE parameter coded on a CREATE override statement. These blocks are contained in the Prime Data Area of the IAM file. Unlike VSAM, IAM files are not limited to block sizes of multiples of 512 or 2048 bytes. They can be created with an optimum block size for the device on which they are being allocated. IAM files typically have block sizes of 1/4 or 1/2 track.

IAM Data Blocks are usually much larger than VSAM CI's, and are not contained in anything equivalent to a CA. However, when loaded, they are used in the same way as a CI. Records are written into each block in key sequence and, if CI freespace was coded on the DEFINE, an equivalent percentage of freespace will be reserved within each block. This freespace is known as *Integrated Overflow*, indicated by the shaded areas below:



As *new* records are inserted into the IAM file after the initial load, they are placed in the appropriate Data Block, using the Integrated Overflow available. This, however, is as far as the similarity with VSAM goes. If the Integrated Overflow has been exhausted in a block, IAM does *not* do a 'split'. Instead, the new record is placed into an Extended Overflow block at the end of the file:



IAM's Extended Overflow Blocks replace the function of VSAM CA Freespace. Unlike CA Freespace, however, Overflow blocks are not tied to specific Prime Blocks. They can be used for any inserts that cannot fit into a Prime Block. They are also structured to cater for variable length records, giving optimum efficiency when the Maximum Record Length for the data records in the file is significantly larger than the defined Average Record Length.

All of this means that less DASD space is required to handle random insert activity when compared to VSAM. When a VSAM file is converted to IAM format, its size can be significantly reduced. Typically, the size of a VSAM file can be reduced by 20-40% simply by converting it to IAM format.

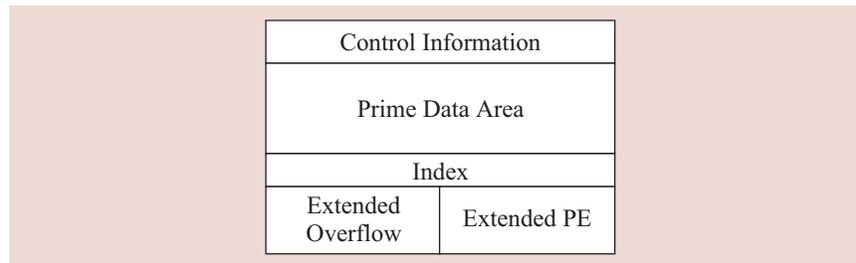
PART THREE

IAM

Inserts to the *end* of an IAM file (*i.e.* new high keys) go into Extended PE blocks. They are similar to Extended Overflow blocks.

An IAM file is 'self-describing'. Its control information (*e.g.* Blocksize, RECFM, LRECL) is stored at the front of the file, followed by the Prime Data Area which contains data loaded by the initial REPRO into the file. The Data Blocks in the Prime Data Area may have Integrated Overflow freespace reserved in them to cater for inserted records.

After the Prime Data Area comes the Index to the Prime Area. This is an efficient index that is held in storage when the file is opened to reduce DASD I/O when processing the file.



Finally, if the Integrated Overflow in a Prime Data Block has been exhausted, or if records have been added to the end of the file, you will find Extended Overflow and Extended PE blocks at the end of the IAM file. Secondary extents are taken by IAM (if coded) when additional DASD space is required for these overflow blocks.

File Reorganization

Despite IAM's more efficient method of handling inserted records, IAM files still require a reorganization from time to time to move records from the Overflow Blocks (and Integrated Overflow) into the Prime Data Blocks.

This Reorganization can be done using the same IDCAMS REPRO jobs that were used for the equivalent VSAM files. The basis for '*REORG selection*', however, will be the amount of Overflow usage (and resulting secondary extents) rather than the number of CI/CA splits, as under VSAM.

Because of the nature of IAM, IDCAMS REORG's of IAM files will usually run faster than a REORG of the equivalent VSAM file. A feature in IAM called 'Backup Compressed' allows the reorganization of compressed IAM files to take place without needing to de-compress and re-compress the data on backup and reload. This saves a considerable amount of CPU and elapsed time on reorganization—particularly on large IAM files.

If REORG's are still taking too long, however, you may wish to apply more intelligence to your IAM (and VSAM) reorganizations by using the FDRREORG component of the FDR DASD Management System.

As described in the '*FDR Storage Management Family*' Concepts And Facilities Guide, FDRREORG allows for a more intelligent REORG selection of VSAM/IAM files based on user specified criteria (*e.g.* '*CISPLIT > nnn*' for VSAM files and '*ORECS=nnn*' for IAM).

This intelligent selection ensures that only the files that really need REORGing will be reorganized.

Using the Optional 'Data Compression' Feature

Requesting Data Compression can further reduce the DASD requirement for an IAM file. This optional feature compresses the data records in the file, reducing the overall file size by an additional 20-50%. Files exceeding 5 cylinders are automatically compressed and smaller files can be compressed via the `DATA COMPRESS=YES` parameter on a `CREATE` override.

Data Compression also reduces I/O's to the file, which has a knock-on effect on CPU usage because it costs CPU to do I/O's. CPU *is* expended doing the de-compression and re-compression of the data, but IAM's efficient compression routines and the reduction in CPU from the reduced I/O's, usually keeps the overall CPU usage below that of the original uncompressed VSAM file.

Support for z/OS Hardware Compression

IAM also supports z/OS Hardware Compression. With the performance improvements that IBM has made on the data compression instructions on the zSeries processors, IAM customers may realize some CPU time reductions when reading data by using z/OS Hardware Compression.

Warning: The use of hardware compression is not recommended on the older processors, because the additional CPU time is substantial.

Hardware compression is an option that users can select for the desired files through the IAM Override facility described earlier. IAM provides one compression dictionary, which is designed to handle primarily text data, with some numerical data, such as might be found in files containing name and address information.

Users can also create their own compression dictionaries to be used by IAM for selected files. Such dictionaries can provide for a higher degree of compression than would be possible with a generic dictionary. When an IAM file is loaded with a user-provided compression dictionary, that dictionary will be written into the IAM file to insure that the dataset can be successfully processed should some subsequent changes be made to the user provided dictionary.

Space Release

Unused space within an IAM file can be released after the file has been loaded or reloaded. This cures the problem of 'over-allocated' files that we discussed in PART TWO. Because IAM files generally take 30-70% less space than VSAM, if the original space allocations are left unchanged in the `DEFINE`'s during the conversion, the unused space can be released.

IAM first checks to ensure that secondary allocation has been allowed for the file before applying the Space Release feature (to prevent immediate Sx37 abends) and also leaves a percentage of freespace after the Release to avoid immediate Secondary Space allocation.

Support for PAV

IAM provides full support for Parallel Access Volume (PAV)—a feature of some DASD devices which provides for concurrent physical I/O activity against the same physical z/OS DASD volume. Use of this capability can provide substantial improvements in response times for online systems or batch jobs that have heavy concurrent I/O activity.

IAM

'In-Core Index', 'Real Time Tuning' and 'Dynamic Tabling'

So far, we have concentrated on the various features of IAM that reduce DASD Space requirements for a file. We have also shown how IAM's file structure can reduce I/O's and CPU by eliminating CI/CA splits, and how the "Data Compression" and "PAV support" features can further reduce I/O's to assist with the performance of a file.

In addition to all of the above, IAM's '*In-Core Index*', '*Real Time Tuning*' and '*Dynamic Table*' features (described below) can also provide a reduction in I/O's and CPU, thus further improve Elapsed Time or Response Time performance.

In-Core Index

When an IAM file is OPEN'd, IAM will read the entire INDEX of the file into Virtual Storage and all references to the index are made to the 'in storage' copy. This essentially eliminates any I/O to the INDEX. VSAM files that have a high ratio of activity to the index component (compared to the data component) usually yield significant savings when converted to IAM.

Real Time Tuning

Through its 'Real Time Tuning' mechanism, IAM will also dynamically tune the allocation and use of DATA buffers to match the demands of the processing program. IAM continually monitors I/O activity as a file is processed and applies the artificial intelligence concept of 'learning by experience' to the file processing.

This process will, of course, differ depending on whether random or sequential processing is being done. IAM determines, and then uses, the number of Data buffers that will result in the best level of performance for the file. IAM then continually monitors buffer usage and determines from its experience if any additional buffers would decrease real I/O. If so, it acquires them. This dynamic tuning of Data buffers eliminates the need to manually calculate, code, and monitor buffer allocation. The files that need more buffers will get them and those that do not, will not.

For multi-volume files, IAM will also initiate multiple-concurrent I/O's to further improve I/O performance (including file Reorganizations on multi-volume IAM files with FDRREORG).

Dynamic Tabling

IAM optionally 'tables' records read from a file in virtual storage. On random reads, IAM checks to see if the requested record is already in its 'Dynamic Table' and if it is, passes it back to the application, eliminating the I/O to the disk. If the record is not currently in the table, IAM reads it from the file, passes it to the application and tables it for subsequent retrievals. If the record is updated by the application, IAM changes the record in the table and on disk.

Files that gain the most benefit from Dynamic Tabling are those where the same records are read and re-read in a short space of time, and where few are being updated. Small files with high random READ activity effectively become in-core tables under Dynamic Tabling.

A Dynamic Table can be requested for a file at access time via the 'DYNCORE=' parameter on an ACCESS Override statement (see below). This is very flexible, because some files may require a table for certain processing (e.g. during the 'Online' day) and not at other times (e.g. Overnight batch). The Dynamic Table can therefore be requested only when needed.

PART THREE

IAM

In the example below, a Dynamic Table of 1000K has been requested for the file pointed to by the FILE3 DD card in JCL. The file's record size will determine the maximum number of records that can be tabled at any one time.

```
//I AMOVRI D DD *  
ACCESS DD=FILE3, DYNCORE=1000  
/*
```

Note: Reductions in Disk I/O made as a result of the 'In-Core Index', 'Real Time Tuning' and 'Dynamic Table' can also contribute to a reduction in CPU usage and provide additional savings on Batch Elapsed Times and Online Response Times. IAM is designed, however, to use storage as efficiently as possible (above the 16MB line) and goes to great lengths not to trade a reduction in I/O for an increase in paging.

IAM Under CICS

IAM files can be used under CICS systems without requiring any change to IAM or to CICS. The FCT entry for the file can go unchanged. If the file is defined to an LSR Buffer pool, the definition can remain unchanged. Although IAM will not use the LSR buffer pool, it can take advantage of other LSR related CICS services which improve CPU usage. The only potential change to CICS would be a reduction in the size of LSR buffer pools. As files get converted to IAM, less LSR buffers will be required for the remaining VSAM files.

IAM supports the BWO (Backup While Open) service, which is primarily used by CICS and CICS/VR to allow for backing up open files, and then for subsequent forward recovery after the dataset has been restored. BWO data includes three flags, indicating the BWO state of the file, and the 8-byte date and time stamp field, which is used as a file recovery starting point. IAM maintains this BWO data, which can be printed on an IAMPRINT LISTCAT report.

IAM Execution Time Statistics

With the potential DASD, CPU, I/O and Elapsed/Response Time savings that can be obtained from IAM, it is useful to be able to monitor and report on these savings. This can, in part, be achieved by requesting the optional 'IAM Execution Time Statistics' report, produced by coding an IAMINFO DD in the accessing JCL.

Once coded in a JCL step, the IAMINFO DD will cause this statistics report to be produced for each CLOSE of an IAM file in that step. The report will include information such as:

- File Attributes (BLKSIZE, LRECL, RECFM)
- Buffer usage (Number Used, Size, Storage Required)
- File Activity (Reads, Writes, Deletes, Inserts)
- Extended Block usage (Overflow Blocks, Extended PE blocks)

The report will also indicate the size of the IAM file so that, during a VSAM-to-IAM conversion, a clear indication is given of the DASD savings made. It also shows the number of I/O's that were saved due to requests being satisfied by the buffers or Dynamic Table. The report will also indicate if IAM was unable to obtain additional buffers for a file due to upper buffer limits or storage constraints. More detailed reports can also be produced by writing the IAMINFO reports as SMF records, which can then be processed by additional reporting programs supplied with IAM (see 'Identifying Candidates' for IAM Conversion page 26).

A report similar to an IAMINFO can also be produced in real time under CICS or TSO. The IAMXMON transaction can supply information on IAM files currently open to a CICS system.

Special Feature: Record Level Sharing (RLS)

In today's 24 x 7 environments it is no longer viable to restrict a file to the rigid protocol of "CICS-only access" during the day, and "Batch-only access" (*i.e.* one-job-at-a-time) during the evening. Now, to fit increasing amounts of work into decreasing windows of time, data centers need to be able to safely access and update their files, whenever they want and from wherever it is necessary.

IAM's Record-Level Sharing (RLS) feature allows multiple applications (CICS Regions, Batch Jobs, TSO Users etc) to concurrently share an IAM file. Each of the sharing systems can have the file open for either read or update. IAM ensures that the integrity of the shared file is maintained during the concurrent access. Individual records within the file are protected from concurrent update and resultant corruption. IAM provides Journaling and Backout services (see below) to recover the shared file to a consistent point, in the event that one of the sharing/updating applications abends.

IAM support for RLS is being introduced in several distinct phases. Phase 1, available now, includes support for SHAREOPTIONS 3 and 4 within the same MVS system. Multiple jobs, CICS regions, and TSO users running within a single system LPAR can now concurrently (and safely!) update an IAM file.

RLS support is available on KSDS files, Alternate Indexes (AIX files), and ESDS files. Not all IAM files used within a CICS system, or referenced by a batch job, or accessed by a TSO user have to be RLS capable. Is it possible to mix-and-match to suit requirements. No extra JCL or programming is required to implement basic RLS support.

An IAM RLS Address Space controls all I/O to RLS-capable IAM files. This includes the required locking of individual records during the update process, together with some sophisticated 'deadlock protection' routines, which minimize the exposure to deadlock situations between the various sharing systems.

Journaling & Recovery

IAM provides the facility to write BEFORE and/or AFTER journal records, which can be employed in either a "forward" or "backout" recovery of a file:

- A "**backout recovery**" is performed in the event of a failure/abend of a single (updating) job. At the point of failure, the "Before" images of the updated data records are written back to the IAM file, working backwards in time, thus removing all the updates.
- A "**forward recovery**" is used if an older copy of an IAM file has to be restored, perhaps after encountering a media failure, or data corruption, or maybe even in the event of Disaster Recovery. Once the older copy of the file is back on disk, the "After" images can then be re-applied to the file, working forwards in time, thus recreating the updates that had taken place.

These procedures, which have been available for some time in IAM, have been enhanced to accommodate the changes introduced by the RLS support described above.

IAM also features a "Dynamic Job Backout" (DJB) facility, which provides a controlled, automated and fully dynamic "backout recovery" of IAM RLS files. If an updating job step abends, all uncommitted updates (for that job) are automatically removed from all RLS files being updated by that job. This negates the need to run separate, manual, batch-driven recoveries for each affected file.

PART THREE

IAM

Special Feature: Alternate Index (AIX) Support

As mentioned earlier, users of the “IAM/AIX” cost option can also define one or more Alternate Indexes (AIX’s) for the KSDS and ESDS files that they have converted to IAM. The procedure for creating an IAM KSDS with an Alternate Index is the same as for VSAM:

- Define the base IAM file with the DEFINE CLUSTER command (*using the \$IAM identifier*)
- Define the AIX with DEFINE ALTERNATEINDEX
- Build the AIX with BLDINDEX
- Define the PATH with DEFINE PATH

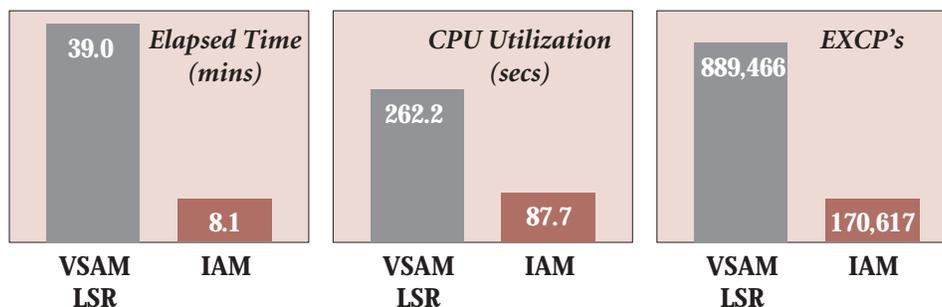
IAM files with AIX’s provide the same sort of savings over VSAM:

- IAM/AIX uses 20-50% less CPU than VSAM
- IAM/AIX performs 50-80% less EXCP’s than VSAM
- IAM/AIX runs in 50-80% less Elapsed Time than VSAM

Here are some results from recent benchmark tests, comparing an IAM Alternate Index with a VSAM AIX that had been tuned with LSR.

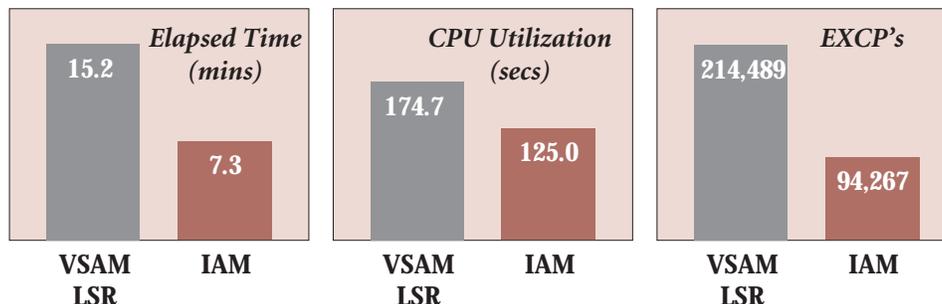
The first benchmark was a **RANDOM I/O** test against the files. This involved doing 256,000 Random READs, 160,000 Updates (half of which required an update to the other Alternate Index), 80,000 inserts, 16,000 deletes, 16,000 Points (start browses) with 80,000 sequential READs (5 records read per start browse).

Notice how IAM took just 8 minutes Elapsed Time compared to VSAM’s 39 minutes!



The second test was a **SEQUENTIAL READ**. This involved reading through the whole of the file, using the NONUNIQUE key Alternate Index.

Notice how IAM took just 7 minutes Elapsed Time compared to VSAM’s 15 minutes !



PART THREE

IAM

ESDS Support

ESDS files are also supported by IAM. Conversion of a VSAM ESDS into IAM is achieved in the same way as a KSDS-to-IAM conversion—using the \$IAM identifier within the IDCAMS NONINDEXED define job.

Two examples of defining IAM ESDS files are shown here:

<u>The 'OWNER FIELD' Method</u>	<u>The 'FILENAME' Method</u>
<pre>//DEFVSAM EXEC PGM=I DCAMS //SYSPRI NT DD SYSOUT=* //SYSI N DD * DEFINE CLUSTER (NAME(AN. ESDS. DATASET) - VOLUMES(MVSO01) - NONI NDEXED - OWNER(\$IAM) - TRK(45 10) - RECORDSI ZE(256 256) SHAREOPTI ONS(2, 3)) /*</pre>	<pre>//DEFVSAM EXEC PGM=I DCAMS //SYSPRI NT DD SYSOUT=* //SYSI N DD * DEFINE CLUSTER (NAME(A. \$IAM ESDS. DATASET) - VOLUMES(MVSO01) - NONI NDEXED - TRK(45 10) - RECORDSI ZE(256 256) - SHAREOPTI ONS(2, 3)) /*</pre>

IAM provides the same advantages for ESDS files as highlighted previously with IAM KSDS and IAM/AIX datasets, namely; reduced DASD space utilization, reduced I/O and reduced batch elapsed times and online response times. These savings are achieved with the same facilities (Data Compression, advanced file structure, Real Time Buffer Tuning Management etc). IAM ESDS file can also have associated alternate index (AIX) datasets.

The internal structure of an IAM ESDS is similar to that previously described for IAM KSDS's. It also has the ability to provide Overflow space within the file. This space is required if the data in the file is compressed. There may be occasions where records increase in size due to a different compression after the update.

Because of this utilization of overflow within the IAM ESDS, the file will from time to time require an IDCAMS Reorganization, similar to that for KSDS's. The process of doing this Reorganization is the same as for KSDS—either a REPRO out to a sequential file and then back into the original file, or a simple REPRO out to a new file. Alternatively, as for real KSDS's and IAM KSDS's, an IAM ESDS can be reorganized with Innovation's FDRREORG as described in the '*FDR Storage Management Family*' Concepts and Facilities Guide.

IAM ESDS datasets can exceed 4 gigabytes as long as the application does not have a dependency on the RBA's (Relative Byte Addresses) of each record being the identical value as VSAM's. This capability is provided by the PSEUDORBA keyword on an IAM CREATE Override when the file is defined or loaded. This indicates to IAM that it can generate a 4-byte RBA that is different than the one VSAM would use for the same record. In effect, IAM will be returning a 4 byte relative record number.

Note: For applications that require a true 4-byte VSAM RBA value, such as the SAP product, the size limitation is still 4 gigabytes of user data.

IAM also supports 8-byte RBA values, also known as Extended Addressability, which was introduced with DFSMS 1.5. This allows ESDS files to exceed 4 gigabytes of user data while using a VSAM compatible RBA value. The IAM support for Extended Addressability does not require DFSMS 1.5, nor does it require the dataset to be SMS managed. This support is triggered by specifying the XESDS keyword on the IAM CREATE Override.

PART THREE

IAM

Identifying Candidates for IAM Conversion

There may also be some files which, although eligible for conversion to IAM, may only be very small or have a very low activity against them. These files would not yield any major savings when converted to IAM. When looking at implementing IAM, it is usually preferable, therefore, to target the eligible files that are the biggest and/or have the highest activity against them, before looking at the smaller and/or lower activity files.

It is not uncommon for just a few files in an application to be responsible for a very high percentage (e.g. 80-90%) of the DASD Usage and I/O Activity. For these reasons, a facility is provided to help identify the files which are both *eligible* for conversion to IAM, and which will offer the greatest savings.

Information about VSAM file size and activity is already being recorded via the standard SMF recording function of MVS (record type 30 subtype 4, and record type 64). IAM utilizes this information and provides a comprehensive reporting utility (IAMSMFVS) to produce a list of the BIGGEST and MOST ACCESSED eligible files. IAM can also supply information on which *jobs/steps* use those files. This information is invaluable when assessing the conversion implications.

IAMSMFVS can be run against the current SMF file or a cumulative history file contained on disk or tape. Here is a sample of the EXCP and SIZE reports that it produces:

<u>VSAM EXCP REPORT</u>										
<u>DATASET NAME</u>	<u>USE</u> <u>CNT</u>	<u>TOTAL</u> <u>EXCPs</u>	<u>RECORDS</u>	<u>READ</u>	<u>INS</u>	<u>UPDT</u>	<u>DEL</u>	<u>SPLITS</u> <u>CI</u> <u>CA</u>		<u>ALLOC</u> <u>TRKS</u>
TEST.FILE	426	28859815	28859815	-	-	-	-	-	-	-
TEST.FILE.INDEX	426	19809594	495	0	0	0	0	30	6	1
TEST.FILE.DATA	426	9050221	469333	57041298	30177	55042	28009	1574	27	7350
CICS.FILE	51	10329082	-	-	-	-	-	-	-	-
CICS.FILE.DATA	51	8516651	252100	5267982	670	890150	42	45	3	5220
CICS.FILE.INDEX	51	1812431	310	0	0	0	0	0	0	15

<u>VSAM SIZE REPORT</u>										
<u>DATASET NAME</u>	<u>ALLOC</u> <u>TRKS</u>	<u>TOT</u> <u>EXCPs</u>	<u>USE</u> <u>CNT</u>	<u>AVE</u> <u>XTs</u>	<u>KEY</u> <u>LRCL</u>	<u>LRCL</u>	<u>BLKSZ</u>	<u>LEN</u>	<u>RKP</u>	<u>CISIZE</u>
BIG.FILE	37155	2507803	24	-	-	-	-	-	-	-
BIG.FILE.DATA	37100	2105001	24	5	223	580	4096	28	0	4096
BIG.FILE.INDEX	55	402802	24	1	-	1529	1536	28	0	1536
PROD.FILE	16540	679216	159	-	-	-	-	-	-	-
PROD.FILE.DATA	16500	270501	159	2	208	208	8192	9	0	8192
PROD.FILE.INDEX	40	408715	159	1	-	4059	4096	9	0	4096

PART THREE

IAM

Another utility (IAMSIMVS) can then be used to simulate the conversion of an eligible file into IAM format to obtain a very accurate estimate of the DASD savings that can be made, both in uncompressed and compressed format. It does this by reading a sample of the records in the file (default 10%) and running them through its compression routines. The simulation is usually accurate to within 5%:

<u>DATASET NAME</u>	<u>VSAM TRKS</u>		<u>IAM TRKS</u>		<u>%SAVINGS</u>	
	<u>ALLOC</u>	<u>USED</u>	<u>STD</u>	<u>COMP</u>	<u>STD</u>	<u>COMP</u>
BIG.CLUSTER	37155	37155	27855	15600	25	58
CICS.FILE.MASTER	21000	19005	12720	9495	33	50
NAME.ADDRESS.FILE	9315	8985	6465	1875	28	79

SMS Support

IAM files can be placed on DFSMS controlled volumes and are treated as non-VSAM datasets. As such, they are cataloged as non-VSAM and have an NVR in the VVDS. From then on, they are treated in the same way as any other VSAM or Non-VSAM SMS file, as follows:

- When allocated, they go through the ACS routines and SMS Volume Selection
- They are managed as per the criteria of the Management Class to which they have been assigned
- They can be backed up, scratched or migrated by the SMS System's Data Manager (e.g. Innovation's ABR or IBM's DFHSM)

An additional benefit under SMS Systems is provided via the DATA CLASS facility of SMS. If a VSAM file is created with a DATA CLASS name containing \$IAM, the DEFINE will be intercepted by the VIF in the usual way and an IAM file will be created. SMS Data Classes can make it simpler to standardize on file attributes such as Blocking Factor, which can be used to improve performance. For this reason, the Data Class method of creating IAM files may be preferable to placing '\$IAM' in the dataset name or Owner field as previously described.

IAM Reliability

As mentioned in the Introduction to this section of the Guide, IAM is a very robust file manager. It has been designed with the aim of preserving file integrity in the event of a system or application failure—even during insert processing.

- IAM relieves the integrity problems caused by VSAM's CI/CA split concept. VSAM clusters can become unusable if an insert is interrupted during a split. Records in the CI or CA being split may be lost if the split does not complete successfully. The structural integrity of an IAM file is always assured. Barring physical damage to the disk, IAM files are logically indestructible. IAM's Overflow concept allows records to be inserted without using splits. Records are added in a single disk write, without the need to update the index.
- IAM provides a recovery program. The 'IAMRECVR' program can be used to read data in an IAM file, even if portions of that file are physically damaged and/or the VIF is inactive. IAMRECVR attempts to recover as much data as possible, copying the undamaged portion of the file to either a sequential file, a new IAM file, or a new VSAM file.

PART THREE

IAM

IAM Installation

Despite the huge benefits that IAM provides over VSAM, it is a very easy product to install. Indeed, it takes only three steps:

- Unload (with IEBCOPY) an Installation Control Library (ICL) from the distribution tape.
- Using the sample 'INSTALL' job from the ICL, load the IAM modules from the tape into an APF authorized library.
- Start the VIF using the sample 'VIFSTART' job from the ICL.

These steps can easily be completed in *less than an hour* and, once done, IAM files can immediately be created by running IDCAMS DEFINE's with \$IAM either in the filename or owner field.

As well as the printed IAM manual that accompanies the product tape, there is also a CD-ROM provided which contains a copy of the manual in PDF and BookManager formats.

IAM ISPF Panels

IAM also comes with a full set of ISPF panels to assist with the running of utility functions against IAM and VSAM files (e.g. Allocation, Delete, Copy, Rename). The report programs highlighted above (IAMSMFVS, IAMSIMVS, IAMXMON etc) can also be invoked from the panels. Tutorials are also included.

Summary

IAM can provide significant savings over VSAM KSDS's and ESDS's in the following areas:

- DASD Space
- I/O's
- CPU

These savings result in:

- Reduced Batch Elapsed times
- Reduced BACKUP/REORG times
- Reduced Online Response times
- Improved DASD utilization

With IAM, you can be safe in the knowledge that your application data is being stored and used as efficiently as possible, with the minimum of resources required (DASD, CPU, I/O). You will be relieved of the effort required to manually tune, implement and monitor your Buffer allocations and the exposure to getting those calculations wrong. Your online system should perform better and will be available each morning, because your overnight batch times will be reduced to allow you to complete jobs in the window available.

IAM is the only real solution to your VSAM problems, because it addresses the root cause—the VSAM file structure—rather than trying to work around it.

APPENDIX

APPENDIX

This is a list of vendor products that have successfully converted their VSAM files to use IAM

ADP	- Paisy
Alltel	- Financial Applications
American Mgmt Sys	- ACAPS, Advantage Financial, HR, Materials Mgmt
American Software	- Accounts Receivable
Anderson Consulting	- DCS
Automated Financial System	- Commercial Loan
BCBS Arkansas	- FISS medicare a payor system
Beta Systems	- BETA93
BMC	- Mainview, VRU Control-M
Bonner&Moore	- Compass
Certegy	- The Collection System
CGI	- CLS
CHECKFREE	- PEP+
Complex Systems	- Banktrade, Trade Finance
Computer Associates	- CA1, CA-7, CA-11, CTMSTATS, Infopoint Tracker, DDA Suite Interest
Compuware	- Abendaid/Fix
Credit Card Software	- Banking Application, ISD, CV, CARDPAC
CSC	- CAPS-I-L
Cyberlif	- Cyberlife
Diebold	- One-Link
Equifax	- Bankcard System
Fair Isaacs	- Triad
First Bankcard Sys	- TBS, CCPS ,BankcardSystem, TCS
GEAC	- Mortgageserv, IQ Connect, MSA/DBS (Financial, General Ledger), Payroll/DBA
Genesis	- Payroll
Global Software	- Fixed Asseis
GMIS	- GMIS/Claimcheck
Group1	- Code-1/Plus, Mail sort ,
H&W	- SYSM ,Mail ,Wizard
HBOC	- Medipac
Hogan	- Hogan Banking Package
Horisonten AB	- Horisonten
IBM	- CICS DFHCSO, SMP/E, Tivoli OS/390, ALCS, SLR, RMDS
Info USA	- White Pages
Information Builders	- Focus
Insystems	- Capsil
Integral	- Intergral PR
ITS	- Blue Shield ITS
Landmark/ASG	- TMON, Monitors,TMON/MVS, TMON/CICS
Levi Ray & Shoup	- Pagecenter

APPENDIX

LIDP	- Administrator
Macro 4	- TraceMaster
Mastersort International	- Scrubmaster
McKinney Systems	- MCI
Metavante	- Custom Statement Formatter
Mobius	- Infopac, ViewDirect
MSI Business Solutions	- NADISPOST
PAISY	- Payroll System
Paysys	- Whirl, Vision+
Pitney Bowes	- Finalist, Mailers Choice
Policy Mgmt Systems	- BCMS, PMS
SAP	- Financial, SAP R2
SCT	- Student Financial Service, SCT Plus
SEMA	- Cardlink
SHAW Systems	- Online Collection
Siemens/Medical Systems	- Invision, EAD/LCR, Hospital billing, Signature
Smartstream Technologies	- Corona
SPI	- SPITABT
Sungard	- INVEST/1, ABC (General Ledger), Omni Plus
Systematics Alltel	- IM, ST, RM
Teracloud	- Spacefinder Workbench
ToneSoftware	- OMC-Report
Trizetto Erisco	- Claimfacts
Unitech Europe	- XACT
Vision Plus	- VP
Walker Interactive	- AP/AR, HR CL

Note: Numerous customers have also successfully converted their own "home-written" VSAM applications to use IAM files.



Corporate Headquarters

Innovation Plaza
275 Paterson Avenue, Little Falls, New Jersey 07424-1658
Tel: (973) 890-7300 Fax: (973) 890-7147
support@fdrinnovation.com sales@fdrinnovation.com
www.innovationdp.fdr.com

European Offices

FRANCE

191, avenue Aristide Briand
94230 Cachan

Tel: (33) 1 49 69 94 02
Fax: (33) 1 49 69 90 98
frsales@fdrinnovation.com
frsupport@fdrinnovation.com

GERMANY

Orleansstraße 4a
D-81669 München

Tel: 089-489 0210
Fax: 089-489 1355
desales@fdrinnovation.com
desupport@fdrinnovation.com

NETHERLANDS (*& Nordic Countries*)

Brouwerstraat 8
1315 BP Almere

Tel: 036-534 1660
Fax: 036-533 7308
nlsales@fdrinnovation.com
nlsupport@fdrinnovation.com

UK

Clarendon House
125 Shenley Road
Borehamwood, Herts
WD6 1AG

Tel: 0208-905 1266
Fax: 0208 905 1428
uksales@fdrinnovation.com
uksupport@fdrinnovation.com

C O N C E P T S & F A C I L I T I E S G U I D E