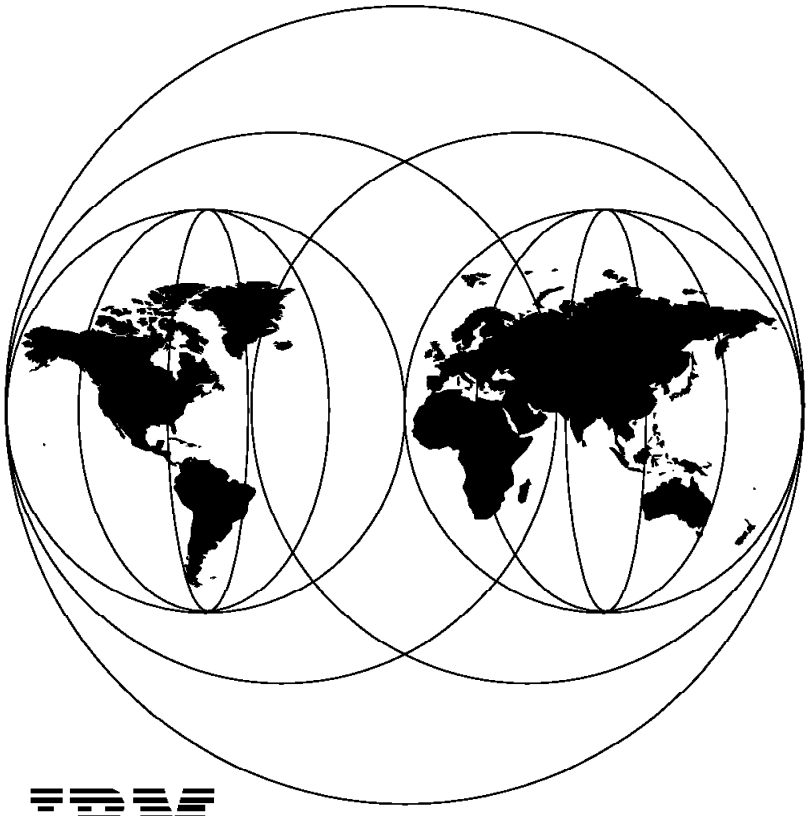
SG24-4301-00

IMS Fast Path Solutions Guide

September 1997



IBM
International Technical Support Organization
San Jose Center

International Technical Support Organization

SG24-4301-00

IMS Fast Path Solutions Guide

September 1997

Take Note!

Before using this information and the product it supports, be sure to read the general information in Appendix D, “Special Notices” on page 145.

First Edition (September 1997)

This edition applies to Version 5 of IMS/ESA, Program Number 5695-176, for use with the MVS/ESA or OS/390 Operating Systems.

Comments may be addressed to:
IBM Corporation, International Technical Support Organization
Dept. QXXE Building 80-E2
650 Harry Road
San Jose, California 95120-6099

When you send information to IBM, you grant IBM a non-exclusive right to use or distribute the information in any way it believes appropriate without incurring any obligation to you.

Contents

Figures

Tables

Preface

IBM′s Information Management System (IMS) Fast Path provides a rich set of facilities for applications where performance, capacity and availability are paramount. This redbook provides an introduction to IMS Fast Path for both the IMS user and the IMS Database Control user who is using CICS as the transaction manager.

This redbook was written for systems programmers, database administrators, and application developers who would like to understand more about the facilities provided by IMS Fast Path. This will give you extra options for increasing the performance and availability of your applications.

A fully functional application system is described in the redbook, with practical examples demonstrating how to define IMS fast-path databases, program specification blocks, and programs to load and access these databases. The full source code is included in the book, so you can use it as a working sample system.

Some knowledge of IMS and application development is assumed.

The diskette accompanying this redbook contains all the source code referenced in book. An HTML version of this redbook is also available on this diskette, in the file “a:\HTML\FP.HTM.”

The Team That Wrote This Redbook

This redbook was produced by two specialists, one from Japan and one from England, while working at the International Technical Support Organization San Jose Center.

Kazutaka Higashi is a Senior Advisory IT Specialist at the High Availability Technical Center in IBM Japan. He has 14 years of experience in the IMS field, specializing in quality verification of IMS to support banking on-line systems those are mission critical with extremely high volume, 100% availability and continuous operation. He holds a degree in mathematics from Waseda University in Japan. His areas of expertise include IMS/FP and XRF. He has written extensively on achieving high availability, continuous operation, and high performance.

Andrew Wilkinson is an IMS systems programmer with IBM United Kingdom. He has 13 years experience with the IMS, in DB/DC application programming, database design, systems programming and occasional consultancy. He has also designed and written extended terminal option exits.

Geoff Nicholls is an Advisory Systems Engineer at the International Technical Support Organization, San Jose Center. Geoff has a Science degree from the University of Melbourne, Australia, majoring in Computer Science. He worked as an application programmer and database administrator for several Insurance companies before specialising in database and transaction management systems with Unisys and IBM. Since joining IBM in 1989, Geoff has worked extensively with IMS customers in Australia and throughout Asia, mostly with Fast Path.

Thanks to the following people for their invaluable contributions to this project:

Attila Fogarasi
Specialist for IMS at the Application Development
and Data Management ITSO Center, San Jose

Tom Ramey
Frank Ricchio
Jack Wiedlin
IBM Santa Teresa Lab

Ron Barber
IBM Almaden Research Center

Comments Welcome

Your comments are important to us!

We want our redbooks to be as helpful as possible. Please send us your comments about this or other redbooks in one of the following ways:

- Fax the evaluation form found in “ITSO Redbook Evaluation” on page 163 to the fax number shown on the form.
- Use the electronic evaluation form found on the Redbooks Web sites:

  For Internet users `http://www.redbooks.ibm.com`
  For IBM Intranet users `http://w3.itso.ibm.com`

- Send us a note at the following address:

  `redbook@vnet.ibm.com`

Chapter 1. An Introduction to IMS Fast Path

This chapter provides a short description of Fast Path for readers who are familiar with IMS but not with Fast Path. Readers familiar with Fast Path may safely skip this chapter. We revisit many of the themes in this chapter in Chapter 5, “When and How to Use Fast Path” on page 47.

1.1 What Is Fast Path?

“Fast Path” is the name of some functions of IMS. These functions were once a separately priced feature, but are now part of the IMS base. Fast Path was developed because existing IMS applications found it hard to provide top performance.

Fast-Path functions fall into two categories, database and expedited message handling (or EMH). These functions are largely independent of each other so we deal with them separately.

1.2 Fast Path Databases

Fast Path databases provide higher performance than ordinary data language interface (DLI) databases. They do not have some of the functions of ordinary DLI databases. Therefore ordinary DLI databases are sometimes called *full-function* databases (or FF databases for short). This name is misleading, because Fast Path databases have some useful (and unique) functions.

There are two kinds of Fast Path database: data entry databases (or DEDBs for short) and main storage databases (or MSDBs for short).

1.2.1 Data Entry Databases (DEDBs)

DEDBs are similar to hierarchic direct-access method (HDAM) databases. IMS uses a randomizer to find root segments and can immediately reuse any free space created when it deletes segments.

Unlike other databases

- DEDBs have an internal structure called a *unit of work*, or UOW (see 1.2.1.2, “Unit of Work” on page 2).
- You can partition a DEDB into many parts (see 1.2.1.3, “Areas” on page 2)
- DEDBs can tolerate many input/output (I/O) errors (see 1.2.1.4, “Multiple Area Data Sets and Record Deactivation” on page 3).
- DEDBs have a segment type optimized for fast insertion (see 1.2.1.5, “Sequential Dependent Segments” on page 3).
- DEDBs have special features for long twin chains (see 1.2.1.6, “Subset Pointers” on page 4).
- You can reorganize a DEDB while it is in use (see 1.2.1.7, “Online Reorganization” on page 4).
- There are dedicated tasks to write DEDB updates to DASD (see 1.2.1.11, “Asynchronous Writes (Output Threads)” on page 5).
- You can load your DEDBs into storage (see 1.2.1.9, “Virtual Storage Option (VSO)” on page 4).

This section concentrates on the difference between DEDBs and other databases.

1.2.1.1 Restrictions on DEDBs

IMS places a number of restrictions on DEDBs. These are mainly to ensure faster performance than full-function databases.

- You can only access a DEDB through an IMS control region (no DLI batch processing). This restriction also prevents a customer-information control system (CICS) local DLI application from using Fast Path.
- DEDBs must be use the IBM Virtual Storage Access Method (VSAM).
- A DEDB can only have 127 segment types.
- DEDBs may not use multiple data set groups (but see 1.2.1.3, “Areas”).
- DEDBs use control interval (CI) rather than segment-level locking. This is not as bad as it might seem, because there is only one anchor point for each CI in a DEDB. There might still be more than one root in a CI, depending on how good your randomizing routine is.
- You have less control over which pointers IMS will store in your segments. We pick this point up later in 5.2.7.4, “Segment Pointers” on page 58.
- You cannot use indexes or logical relationships with DEDBs.

  This last is the most severe of the DEDB restrictions. You can overcome it by writing your application to implement indexes or logical relationships. These functions are costly to provide, so you may find that your application performance suffers as a result.

1.2.1.2 Unit of Work

A “unit of work” consists of a number of CIs. When you design your DEDB, you decide how many CIs make a unit of work. When you decide how big the various parts of a DEDB should be, you specify the size in units of work.

A number of utilities lock at unit of work level, so it is important not to make the unit of work too large. On the other hand, if your unit of work is too small, the minimum amount of overflow space (one CI) might be too large a proportion of the unit of work. There are more details on how to choose the size of a unit of work in 5.3.3, “Picking a Unit of Work Size” on page 62.

1.2.1.3 Areas

You can divide your DEDB into a number of data sets, called *areas*. These are not the same as data set groups, because an area contains all segment types. Most commands and utilities operate at an area level, rather than on the whole database. For example you can restore a damaged area of a DEDB while the rest of it is in use.

You will probably want to write your own randomizer for a DEDB with more than one area. If you cannot wait to find out why, turn to 5.2.1.1, “Why You Would Write a DEDB Randomizer” on page 51.

1.2.1.4 Multiple Area Data Sets and Record Deactivation

You can have up to seven copies of each area data set. These are called multiple area data sets (or MADS for short). You can have a total of 240 area data sets in all.

IMS reads from only one area data set at a time, but uses them all in turn, thus spreading the I/O load across several devices (if that is how you have defined your data sets). If IMS cannot read a CI from one area data set, it will read the CI from another area data set. IMS remembers that the read failed and will not try to read that CI from that area data set. This is called *record deactivation*.

IMS writes to every area data set, in order to keep them all in line. If IMS cannot write a CI to one area data set, it deactivates that record. If IMS cannot write a CI to any of the area data sets, it prevents further access to that CI, but keeps the area open.

Provided you have at least one good copy of each CI, IMS can always create an error-free area data set. You can do this while the area is still in use.

IMS can keep track of up to ten bad writes and four bad reads per area data set. IMS stops using an area data set if it exceeds these limits. This is not a disaster if you are using MADS. You can defer database recovery until a convenient time. However, we do not recommend that you run for a long period of time with bad CIs.

There is a similar feature for full-function databases, but the I/O error tolerance for DEDBs is much more sophisticated.

There is more about record deactivation in 2.1.6, “DEDB Record Deactivation” on page 11.

1.2.1.5 Sequential Dependent Segments

A sequential dependent segment (SDEP) is a special kind of segment which is very quickly inserted. The restrictions on this kind of segment stem from that.

There can only be one SDEP segment type in a DEDB and it must be the first child of the root in the DBD. If you want to process the segments in a different order, you can change the order of the segments in your PSB.

An SDEP cannot have child segments.

IMS holds all SDEPs together in a special part of the DEDB. It holds the SDEPs in the sequence they were inserted, regardless of which root they belong to. This reduces the number of write I/Os.

SDEPs still exist even if you delete their root. In this case, you cannot retrieve the SDEPs except by using the utilities.

You cannot use data sharing with SDEP segments in IMS Version 5.

IMS maintains a last-in, first-out (LIFO) pointer chain through the SDEPs of each root. This is so that IMS can insert the next segment with a minimum of I/O, but it also means that an SDEP cannot have a sequence field. Although it is possible to follow the pointer chain to retrieve a given SDEP, doing so is likely to involve a lot of I/O. The SDEPs of a root will be scattered throughout the DEDB, so each pointer is likely to point to a different CI.

There are utilities to read and to delete SDEPs. These utilities process SDEPs in insertion sequence and so cannot tell which root a given SDEP belongs to.

There is a special DLI call (POS) which finds the position of SDEPs. This is useful when deciding where one of the batch utilities should start.

The collective term for non-SDEP segments is *direct dependent* segments, sometimes shortened to *DDEP* segments.

1.2.1.6 Subset Pointers

Subset pointers (also known as *SSPs*) are a special kind of pointer, which your application can set as well as use. You can have up to eight subset pointers per segment, but they are not allowed on the root or on SDEPs.

Typically you would use a subset pointer to keep your place in a long twin chain, so that IMS would not have to search through the chain from the beginning.

You set and use subset pointer with special SSA command codes.

We go into more detail about subset pointers in 2.1.3, “Subset Pointers (SSP)” on page 10 and 5.2.6, “Using Subset Pointers” on page 56.

1.2.1.7 Online Reorganization

The high-speed DEDB direct reorganization utility reclaims fragmented space while the database is still online. It operates on a unit of work level. When IMS creates a DEDB area, it reserves a unit of work for this utility. The utility locks one unit of work at a time, and builds a reorganized copy in the reserved unit of work.

Note: You cannot use this utility to change the structure of a DEDB.

1.2.1.8 Position and Path Calls

IMS automatically uses multiple positioning without requiring you to declare it in your program specification block (PSB). This can confuse you if you are not expecting it. Similarly, IMS automatically allows path calls for DEDBs, without your having to declare it in your PSB.

1.2.1.9 Virtual Storage Option (VSO)

This is a special kind of DEDB, new with IMS Version 5. IMS keeps VSO CIs in a data space after it has read them. IMS does not go back to the direct-access storage device (DASD) when it needs to read a CI again. IMS does not keep VSO SDEP CIs in the data space (on the assumption that you are going to read them in a batch with a utility).

VSO databases have better locking mechanisms than other DEDBs. IMS locks VSO data at segment level rather than CI level. IMS releases locks on VSO data once it has written the data to the data space. This is a faster process than with other DEDBs, where IMS holds the locks until it has written the data to DASD.

You have the option to open VSO DEDBs or even to load entire VSO DEDBs into storage when IMS starts.

There are some restrictions on data sharing with VSO.

VSO DEDBs are similar to MSDBs. The IMS developers intend VSO to replace MSDB (see 1.2.2.4, “Migrating Your MSDBs” on page 7 for more information).

We have more to say about VSO in 2.1.4, “Virtual Storage Option (VSO)” on page 11 and 5.2.4, “Using VSO” on page 54.

1.2.1.10 High Speed Sequential Processing

High speed sequential processing (HSSP) allows you to process a DEDB much faster than normal, provided you adhere to these restrictions:

- HSSP can be used only by a non-message-driven batch message processing (BMP).
- You must process the database in strict hierarchical sequence.
- You must declare your intention to use HSSP in your PSB.
- HSSP cannot be used twice on the same area at the same time.
- HSSP cannot be used on an area at the same time as a Fast Path utility.

HSSP locks at unit of work level.

Several of these restrictions apply because HSSP uses its own buffer pool (and not the common Fast Path database buffer pool).

You can ask HSSP to take an image copy while it is processing. This is a fuzzy image copy, but it is much faster than running your program with HSSP and then taking an image copy. This is because HSSP is clever enough not to record a log entry if it is also creating an image copy.

HSSP also allows you to process some but not all of the areas of a DEDB.

We have more to say about HSSP in 5.2.5, “Using HSSP” on page 55.

1.2.1.11 Asynchronous Writes (Output Threads)

Asynchronous writing is one of the things that makes DEDBs faster than other databases. During commit processing, IMS waits only for the logging of the DEDB updates to occur. IMS leaves the actual updating of the database to another task called an *output thread*. This allows the transaction to finish and another one to start, thus improving performance. However, a new transaction would have to wait if it wanted a CI which an output thread was still writing to DASD.

This is in contrast with full-function databases, where IMS waits for both the logging and the database updates before proceeding.

1.2.1.12 Fast Path Database Buffers

Fast Path databases do not use the DLI buffer pool. Instead, they have their own buffer pool. IMS expects this buffer pool to be small and reuses buffers at the earliest opportunity.

One of the effects of this is that IMS does not remember which CIs are in the buffer pool unless they are currently being used. This is in contrast to full-function databases, where IMS will leave CIs in the buffer pool as long as possible. We have more to say about this in 5.1.2, “Holding Data in Main Storage” on page 49.

Each region has a number of buffers dedicated to it. A small number of overflow buffers is also available for all regions to share. If a program exceeds its allocation (including overflow), IMS backs out all its updates, and abends the program if it is a message program. We have more to say about this in 5.1.3, “Fast Path Database Buffers and Output Threads” on page 50.

IMS never writes Fast Path buffers before it logs the updates. This means that DEDB updates never need to be backout out, although they may still need forward recovery.

1.2.1.13 Field Call Use

You can use field calls with DEDBs. These give improved locking and are described in 1.2.2.3, “Field Calls.”

1.2.2 Main Storage Databases (MSDBs)

An MSDB is a kind of database that is always in storage and is therefore very quickly accessed. There are four different kinds of MSDB, with different characteristics. We will not go into these differences here because there will be no further development for MSDBs. In fact, IMS developers intend to remove support for MSDBs at some point in the future. See 1.2.2.4, “Migrating Your MSDBs” on page 7.

1.2.2.1 MSDB Restrictions

IMS places several severe restrictions on MSDBs:

- You cannot access an MSDB unless IMS is up (because MSDB are held in storage).
- You cannot access MSDBs from a CICS region, even if you use DBCTL.
- MSDB segments must have a fixed length.
- MSDBs can only have root segments, and so cannot have indexes or logical relationships.
- MSDBs cannot have multiple data set groups (there are no data sets).
- You cannot use field-level sensitivity with MSDBs.
- You cannot use IMS segment search arguments (SSA) command codes or Boolean operators with MSDBs.

1.2.2.2 MSDBs and DASD

MSDBs are not lost while IMS is down. IMS reads MSDBs from DASD at start up, then writes them back to DASD at system checkpoints and at closing.

There are utilities to help you manipulate the DASD copies of MSDBs.

1.2.2.3 Field Calls

These are a special kind of DLI call that you can use for highly active segments where normal locking would slow performance. A field call updates a database at the field level (hence the name). You can use this call with MSDBs and (from IMS Version 5) with DEDBs as well.

A field call causes IMS to lock a segment for a short time during commit processing (and not at all when you make your field call). This helps increase transaction processing, but does mean that the data can have changed between your call and the commit. One of the field-call options is to verify that other updates have not invalidated your change.

You would typically use field calls for a rapidly changing field that many transactions need to access. For example you might use field calls for a process counter for a transaction designed for parallel processing. We explain this example in greater detail in 5.1.1.3, “An Example Using Field Calls” on page 48.

1.2.2.4 Migrating Your MSDBs

From IMS Version 5, all MSDB features are available to VSO DEDBs.

If you are already using MSDBs, you should migrate them to VSO DEDBs. There is a program (called the *MSDB-to-DEDB conversion utility*) to help you migrate. You can migrate without changing your application code.

There are more details on the conversion process in 5.6, “Converting an MSDB to a DEDB” on page 65.

If you are planning a new application you should not use MSDBs. Later releases of IMS will not enhance MSDB support and may drop support for MSDBs altogether.

1.2.3 Mixing Fast Path and Full-Function Databases

It is important to realize that you can mix Fast Path and full-function databases. This may or may not be a bad idea, depending on why you chose to use Fast Path.

1.2.3.1 Speed

If you choose to use Fast Path databases for speed, then mixing them with full-function databases is likely to cause performance bottlenecks. IMS waits until it has written all full-function database updates and the matching log records to DASD before completing the commit process. With Fast Path databases, IMS waits only for the logging to occur. In a very busy system, another transaction could be wanting to use this region, but it must wait for the full-function database updates to complete.[1]

If you already have full-function databases in your application, converting some or all of them to Fast Path will improve performance. This is because output threads handle the Fast Path database updates, leaving the commit process less to wait for. If you have a choice about which databases to convert to Fast Path, choose the databases that you update most often.

1.2.3.2 Function

If, on the other hand, you choose Fast Path databases for their function, then mixing them with full-function databases has no effect.

If you already have full-function databases in your application, converting some or all of them to Fast Path could, for example, provide better I/O error tolerance.

1.3 Expedited Message Handling (EMH)

EMH is a faster method of handling messages because it does not use the IMS message queue, or the usual scheduling algorithms. EMH uses a special type of region, called a *Fast Path* or *IFP* region.

Please note that you can use the Fast Path databases without using EMH. You should use EMH only where performance is so critical that even the few I/Os associated with the IMS message queue are too many. If you use EMH with a

[1] This problem does not arise if you only read the full-function databases, as there are no updates to log and so no effect on performance.

transaction that does a lot of database input and output, any improvement in message handling may be too small to notice.

You can also use EMH without using Fast Path databases. This is unlikely to improve performance unless the processing does not involve any databases at all.

EMH was written at a time when the usual message-queue handling was less efficient than it is now. EMH is faster than using the message queue, but the difference is less than it used to be.

We expand this topic in 5.7, “Using the Expedited Message Handler” on page 65.

1.3.1 EMH Restrictions

IMS stores EMH messages in special buffers in the extended common service area (ECSA). Only one buffer per terminal can use EMH. (With ETO, there is one buffer for every ETO terminal that is currently using EMH.) These restrictions keep ECSA usage to a minimum:

- Input messages must have a single segment, be in response mode, and come from a terminal, not a program or a multiple systems coupling (MSC) link. This is to prevent more than one message segment being queued to or from a terminal at any one time. It also keeps you from using conversational transactions, because using a scratch pad area (SPA) makes the message multisegment.
- Output messages must be single segment, and sent to the originating terminal.
- IMS discards a user’s message if there is no Fast Path region running that can process it. To prevent this, IMS supplies a user exit called the *Fast Path input edit/routing exit (DBFHAGU0)* that allows you to route an EMH message through the usual message process to a message processing program (MPP) region. We have more to say about this exit in 5.7.2.3, “Fast Path Input Edit/Routing Exit (DBFHAGU0)” on page 67.
- You cannot use IMS command calls with EMH.
- IMS processes EMH messages on a first-in, first-out basis; it make no attempt to prioritize messages on other criteria.
- IMS fast-path regions must wait for input, so they can contain only one application program. You can overcome this restriction by writing the program to call a different process, depending on the contents of each message.

1.4 Fast Path and CICS

If your application runs under CICS you can use DEDBs, provided you also use database control (DBCTL). However, if you use the CICS local DLI, your application is essentially the same as batch DLI and such applications cannot use DEDBs.

You cannot use main storage databases from a CICS application. EMH is not relevant because CICS has its own scheduling methods.

Chapter 2. Major Fast Path Features

Fast Path has added many attractive new features since it was first introduced with IMS/VS 1.3. This chapter summarizes the newer Fast Path features under the following headings:

- Data entry database (DEDB)
- Main storage data base (MSDB)
- Expedited message handler (EMH)
- Application programming interface (API)
- Utilities

2.1 Data Entry Data Base (DEDB)

Before IMS/VS 1.3, a DEDB had several restrictions which limited its use to specialized situations. With the removal of these restrictions, it has became possible to use DEDBs in most applications.

Most of the enhancements have been to provide yet higher performance and higher availability in both normal and error situations. Even today, DEDB data availability remains unmatched by other database and file management products.

The more recent features described in this section are:

- Enhanced DEDB record structure
- Multiple area data sets (MADS)
- High-speed online reorganization
- Subset pointer (SSP)
- Virtual storage option (VSO)
- High speed sequential processing (HSSP)
- DEDB record deactivation
- Data sharing
- DEDB block size
- Expansion of DEDB independent overflow (IOVF)
- Log reduction
- DEDB compression exit

2.1.1 DEDB Record Structure

A DEDB is a hierarchical direct (HD)[2] type of database. It can contain up to 127 segment types, in up to 15 hierarchical levels. The segment types, as shown in Figure 1 on page 10, are:

- A root segment with either

[2] hierarchical direct

– A sequential dependent (SDEP) segment and up to 125 direct dependent (DDEP) segment types, or

– A maximum of 126 DDEP segment types

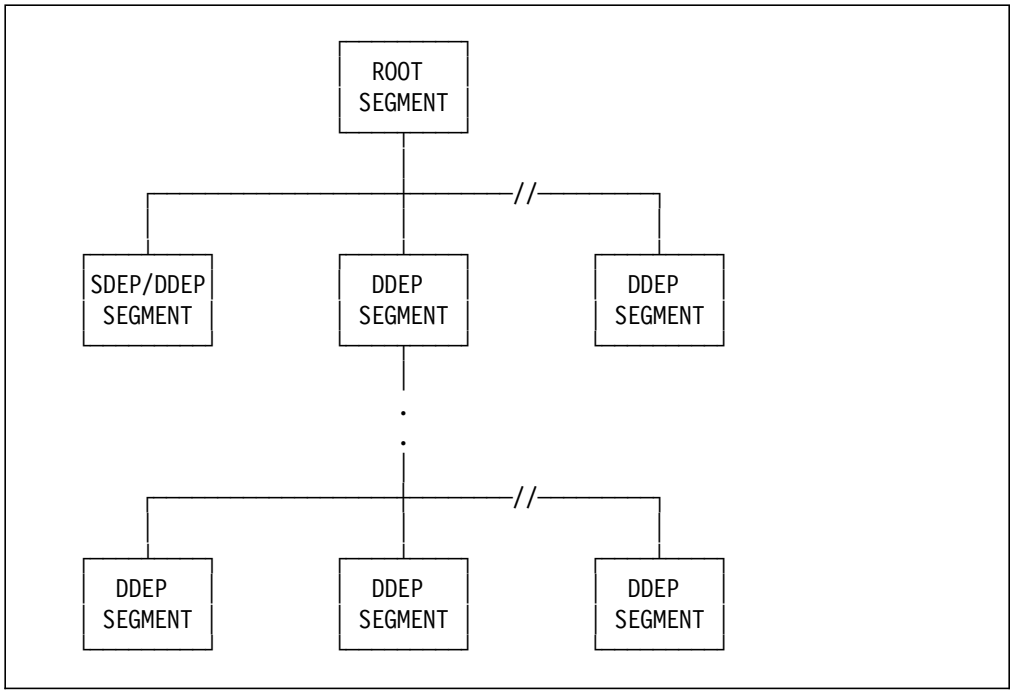


Figure 1. DEDB Record Structure

2.1.2 Multiple Area Data Sets (MADS)

To enhance their availability, DEDB areas can be replicated with a maximum of seven copies per area. Each copy is called an area data set (ADS). The multiple ADSs for an area should be isolated from each other as much as possible. They can be on different types of DASD.

The available copies are used in turn to satisfy read requests, but writes must be done to all copies. At least one ADS per area is required.

Extra ADS copies may be added dynamically using the Area Create Utility.

See 3.1.2, “Multiple Area Data Sets” on page 20 for further information.

2.1.3 Subset Pointers (SSP)

Applications that have to deal with long chains of direct dependent segments can benefit from the subset pointer support introduced in IMS 1.3. The use of subset pointers can reduce the number of unproductive retrieve calls and internal pointer chasing when accessing the last part of a long twin chain. See 4.1.5, “Subset Pointer (SSP)” on page 34 for detailed information.

2.1.4 Virtual Storage Option (VSO)

The virtual storage option (VSO), introduced in IMS Version 5, allows any areas of a DEDB, subject to available real storage, to be loaded into a multiple virtual storage (MVS) data space. There is an implementation option to either read the area into memory at open time or as each control interval is first referenced. Once a CI has been read into memory, all application or utility reads are made from the data space. When a CI is updated, it is written to the data space and the lock is released immediately. Periodically, IMS writes out updated CIs from the data space to the area data sets on the DASD.

VSO will be attractive to customers with areas that are very heavily used, but in addition, it is intended to provide a replacement for the main storage database. For many years, MSDBs have offered exceptionally high performance, including the ability for multiple applications to update the same data at the same time with integrity using the field (FLD) call. However, their use was subject to certain restrictions, such as single segment hierarchy, fixed length segments only, no database recovery control (DBRC) support, and nonstandard backup and recovery procedures and utilities. VSO DEDBs can have the full hierarchy, and use standard DBRC, backup and recovery facilities. To complement the VSO option, DEDBs have generally been enhanced in other ways. Segments can be defined as fixed or variable length, and the DL/1 Field Call (FLD) has been implemented for DEDB. (Note that these apply for all DEDBs, not just for VSO DEDB.)

Additional information can be found in 4.1.6, “Virtual Storage Option (VSO)” on page 36.

2.1.5 High Speed Sequential Processing (HSSP)

High speed sequential processing (HSSP) is a special high-performance option for BMPs that process one or more areas in physical sequence (for example, by repeatedly issuing get next (GN) Root calls). In IMS Version 3 and IMS Version 4, the performance is largely enhanced by exploiting cached DASD controllers. In IMS Version 5, HSSP has been rewritten to use software techniques for gaining even higher performance.

An option of HSSP is to take an image copy while the user application is sequentially processing through an area. In IMS Version 5, several major restrictions were lifted, and the image copy option is now much more usable.

Full details can be found in 4.1.4, “High-Speed Sequential Processing (HSSP)” on page 34 and 3.3.5, “HSSP Image Copy Option” on page 28.

2.1.6 DEDB Record Deactivation

When a write I/O error occurs on a DEDB, the CI concerned ceases to be available. IMS records this fact in the second CI (a control CI), and the rest of the CIs remain available. This is called *record deactivation*. In a MADS environment, the affected CI was probably written successfully to the other ADSs, and so remains available for subsequent readers.

Because information about the error is recorded on the ADS itself, there is no danger of accidentally reading bad data, even after a cold start.

See 3.1.3, “DEDB Record Deactivation” on page 21 for more information.

2.1.7 Data Sharing

Since IMS/VS 1.3, data sharing has been supported for fast path DEDBs. Support is provided in two flavors: area-level sharing and block-level sharing. As with standard DL/I data sharing, each type requires registration with DBRC, and block level sharing requires the IMS Resource Lock Manager (IRLM).

Area-Level Sharing: Area-level sharing is generally equivalent to database-level sharing for a full-function database. That is, DEDB areas can be shared between two or more IMS subsystems, with one system having update (UP) authorization and the others having read-only (RO) authorization, or between multiple subsystems with read (RD) or read-only (RO) authorization. The database, all areas, and all area data sets must be registered with DBRC.

Block-Level Sharing: Block-level sharing for DEDBs is supported for both intrasystem sharing (SHARELVL 2) and intersystem sharing (SHARELVL 3).

With IMS Version 5, DEDBs with an SDEP segment defined cannot participate in block-level data sharing. Areas using VSO are also excluded.

2.1.8 Control Interval Size

Prior to IMS Version 2, DEDB control interval size was limited to 4 KB or less. Previous releases did not allow larger CIs because DEDBs used VSAM Improved Control Interval Processing (ICIP) as their access method and VSAM ICIP has a maximum CI size of 4 KB. This restriction was removed in IMS Version 2 where media manager became the access method for DEDBs instead of VSAM ICIP. The media manager allows DEDBs to use CI sizes up to 28KB in multiples of 4K.

The size and maximum number of fast-path buffers was increased in IMS Version 4. The maximum buffer size (BSIZE) is now 32 KB and the maximum number of buffers (DBBF) is 99,999.

2.1.9 Expansion of DEDB Areas

Prior to this enhancement, if a DEDB area ran out of space, an IMS outage was required to increase the data set size for the area. It is now possible to increase the data set size, and make the additional space available as extra overflow CIs, without stopping IMS. This is explained in 3.1.4, “Increasing the Size of an Area (Expansion of IOVF)” on page 22.

2.1.10 Log Reduction

Fast path has always been very efficient in its production of log data. Fast path database updates are never done before sync point, so there is never a need to back-out updates. Consequently, fast path logs only the after-images of data, rather than the before and after images.

Prior to IMS Version 3, the image of an entire segment was logged when it was replaced, even through much of the data might have remained unchanged. IMS Version 3 introduced log reduction, where only the changed data is logged, rather than the whole segment.

Further details can be found in 4.1.7, “Log Reduction” on page 38.

2.1.11 DEDB Compression Exit

DEDB segment compression has been available since IMS Version 3, using the familiar segment compression exit that has been available for full function databases for many years. This provides the ability to compress, expand, encrypt, or decrypt DEDB segments. Two sample exit routines are provided. These routines are usable with both DEDB and full-function databases. They use compression facilities provided by MVS/ESA. The advantages of using DEDB compression include:

- Reduced DASD requirements for DEDBs
- Potentially reduced I/O, since more data can be placed in fewer CIs

The steps to implement DEDB compression are these:

- Create DEDB compression exit.
- Take an image copy of the DEDB.
- Unload the DEDB by DBTOOLS, using the old DBD.
- Execute DBDGEN and ACBGEN.
- Delete and define the DEDB with a new space allocation.
- Reload the DEDB by DBTOOLS, using the new DBD.
- Take another image copy.
- Restart IMS.

2.1.12 Other IMS Version 5 DEDB Enhancements

Other DEDB enhancements in IMS Version 5 include the option for selected areas to be preopened at IMS start-up time (or /START command). The Q command code can be used by an application to guarantee that no other program will update the CI until the calling program has reached its sync point. The DL/1 DEQ call can now be used with DEDB to invoke buffer stealing, and release CIs locked with the Q command code.

2.2 Main Storage Data Base (MSDB)

A main storage database (MSDB) provides exceptionally high performance. The MSDB realizes its high performance by allowing solely root-only databases with no database I/O, by minimizing resource contention when using the FLD call, and by having a much reduced path length.

Since IMS Version2, virtual storage constraint relief has been provided, but also certain limitations have become apparent.

2.2.1 Virtual Storage Constraint Relief (VSCR)

MSDBs reside in main storage rather than on DASD. Before IMS Version2, the MSDBs were resident in the common storage area. As part of the virtual storage constraint relief (VSCR) in IMS Version 2, MSDBs were moved above the 16 MB virtual storage line into extended CSA (ECSA).

2.2.2 MSDB Limitations

The Announcement Letter for IMS Version 2.2 recommended that customers to use only non-terminal-related MSDBs with user-defined keys but without terminal-related keys. Clearly, the intent was to functionally stabilize the other three types of MSDBs, and the reasons behind this became apparent with the introduction of the extended terminal option feature in IMS Version 4.

Also, when using APPC/IMS in IMS Version 4, MSDBs cannot be updated by transactions from LU6.2 devices. The implied recommendation is to convert all MSDBs to VSO DEDBs in IMS Version 5. However, all four types of MSDB remain available in IMS Version 5.

2.3 Expedited Message Handler (EMH)

The Expedited Message Handler (EMH) provides an alternative to full-function IMS message queuing and application program scheduling. By supporting only a subset of the standard message-processing facilities, EMH is able to process messages with a significantly reduced path length.

In the past few years, EMH has received virtual storage constraint relief, and in IMS Version 4, the management of the terminal buffers was completely revised to use a dynamic buffer pool.

Further information can be found in 4.4, “Expedited Message Handler” on page 44.

2.3.1 EMH Buffer Pool (EMHB)

In IMS Version 4, the previously static EMH terminal buffers became dynamic buffers, managed in a pool that expands and contracts as required. The pool is called the *EMHB pool*. Buffers are acquired when EMH transaction input is received, and freed when no longer being used.

2.4 Application Programming Interface (API)

There are four important enhancements to the application programming interface (API) since IMS Version 1.3:

- Enhanced SSA support
- New language support
- INIT call
- Field (FLD) call for DEDB

2.4.1 Enhanced Segment Search Argument (SSA) Support

The advantages of enhanced SSA support are these:

- Multiple SSAs per Call — A call to a DEDB may now contain as many SSAs as there are levels in the database hierarchy.
- Boolean operators — An SSA may contain the Boolean operators AND and OR. The “INDEPENDENT AND,” which is used only in conjunction with secondary indexing, is not supported.

- Command codes (including Q) — An SSA may contain any of the command codes allowed in standard DL/I database calls. The one exception has been the Q command code, which was enabled for use with DEDBs only in IMS Version 5.

  Use the Q command code if you want to prevent another program from updating a segment before your program reaches a commit point. This means that you can retrieve segments using the Q command code, and then do subsequent processing secure in the knowledge that the buffers have not been released and that no other program has updated those segments.

  Although fast path requires the Q command code to follow the same rules as for a full-function IMS, where it is a two character command code, the second character (lock class) has no significance.

  For full-function databases, when you use the Q command code and the dequeue (DEQ) call against the input-output program communication block (IOPCB), you reserve and release segments by lock class. However, in IMS Version 5, a DEQ call can be issued against a DEDB PCB. Again, the Q command code does not support lock class. A DEDB DEQ call causes Fast Path to invoke buffer stealing and so release any application buffers that satisfy one of the following conditions:

  Buffers that have not been modified

  Buffers that do not protect a correct root position

  Buffers that have been previously protected by Q command codes.

- Path calls

  Before IMS Version 1.3, only one SSA per call was allowed and hence path calls were not supported. However, the PCB processing option which allowed path calls (PROCOPT=P) was used by Fast Path to request a special status code (GC) when a unit of work boundary was crossed by a non-message-driven BMP.

  Since IMS/VS 1.3, the interpretation of PROCOPT=P has remained the same, with GC status code returned to the program when crossing a unit of work boundary. Path calls are always allowed for DEDB PCBs, regardless of the PROCOPT.

2.4.2 New Language Support (PASCAL, C)

Several new languages are supported for application development. From IMS Version 3, the following have been available:

- VS PASCAL
- C/370
- C language

IMS Version 4 added support for:

- REXX — IMS now supports writing application programs in the REXX language. This allows IMS users to interactively develop REXX execs for execution in all IMS region types (MPP, BMP, IFP, or batch).

  The REXX language provides an enhanced application programming environment. It is especially suited for prototyping or rapid development of moderate-volume transaction programs.

Further information can be found in Appendix C, “Using REXX with IMS” on page 139.

2.4.3 Initialization (INIT) Call

The INIT call can be used by IMS online and batch programs after IMS version 2. By using the INIT call, application programs that attempt to access unavailable data receive a status code (BA, BB) that allows them to avoid being pseudo-abended (U3303).

The INIT call that is relevant here uses the STATUS GROUPA character string in the I/O area. This form of the INIT call is used to alert IMS that the application program understands and is prepared to handle the new status codes. The new status codes are BA and BB:

- BA indicates that required data was not available. Either the database was not available or the requested block or record was not available. (A record could be unavailable if the call requests a block-level data sharing lock that is held by a failed subsystem.) Any updates done by IMS as part of this call processing have been backed out. The state of the database is the same as it was before this call was issued.
- BB is the same as BA, except that backout has proceeded as far as the program′s last commit point. The backout is done for all databases. All messages, except those issued through an EXPRESS=YES PCB, have been cancelled. Database position for each PCB is at the start of its database.

  This status code is returned for calls to fast-path databases only. If the call has done no updating before the unavailable data is encountered, a BA is returned. If updating has been done, the BB is returned.

The INIT STATUS GROUPA call should follow the first get unique (GU) command to the I/O PCB in message processing programs (MPPs). If the INIT call precedes the GU, the primary message is lost but is retrieved again when the GU is issued.

2.4.4 Field (FLD) Call to DEDB

The Field (FLD) call is used to change one or more fields in a segment subject to specified fields meeting certain criteria. Both the verify and the change operations are specified in the one FLD call. It is used in place of a GHx and REPL pair of calls, and is more efficient because it uses less CPU, takes a lock only at sync-point time and creates less log data.

The FLD call has always been available for use with MSDBs. In IMS Version 5, it became available for use with DEDBs as well.

The FLD call is more fully explained in 4.2.1, “Field (FLD) Call” on page 40.

2.5 Utilities

Eight utilities relevant to Fast Path have been enhanced since IMS Version 1.3:

- Log recovery utility (DFSULTR0)
- Database image copy utility (DFSUDMP0)
- HSSP image copy utility
- CIC enhancement utility

- DEDB initialization utility (DFSUMIN0)
- DEDB area data-set create utility (DFSUMRI0)
- DEDB area data-set compare utility (DFSUMMH0)
- Online DEDB reorganization utilities

2.5.1 Log Recovery Utility (DFSULTR0)

The log recovery utility was modified in IMS Version 1.3 to handle the new DASD logging feature. When an online IMS system fails, the on-line log data set (OLDS) might not be closed by extended subtask ABEND exit (ESTAE) processing. If so, it is closed by emergency restart or by the log recovery utility. In either case, the WADS is read and used to write the last few buffers to the OLDS prior to closing it.

In IMS Version 1.3 and IMS Version 2, if incomplete chains of Fast Path log records exist in the OLDS, the utility replaces them with padding records (x'48').

Since IMS Version 3, the log recovery utility does not replace incomplete chains of Fast Path log record with padding records, because the database recovery utility applies only complete sets of log records for DEDB recovery. The same is true for emergency restart.

2.5.2 Database Image Copy Utility (DFSUDMP0)

In IMS Version 1.3, the database image copy utility was enhanced to provide support for multiple area data set (MADS) and a new concurrent image copy option was added:

- Multiple Area Data Set (MADS) support — The image copy utility can use MADS input, but only those area data sets (ADSs) with a status of available in the RECON will be opened.

  The utility uses READ ANY processing to read from the various ADSs. If all ADSs have an error queue element (EQE) for the same control interval, or if a read error occurs for the same CI of all ADSs, the utility will terminate. The result of an image copy in this environment will be a good image copy from several ADSs which may have had READ or WRITE errors.

  It should be emphasized that the image copy is a copy of an ***area***, not of an ADS. The second control interval will contain no EQEs, and an ***area*** recovered using this image copy will contain no EQEs.

- Concurrent Image Copy (CIC) option support

  The second change to the image copy utility allows the user to request a concurrent image copy. With this, the user can produce a batch image copy while the area is still online. In IMS Version 5, the CIC option for DEDB was enhanced to remove the previous need to briefly stop and start an area before copying it.

  This is explained in 3.3.4, “Enhanced Concurrent Image Copy (DFSUICP0)” on page 28.

2.5.3 High-Speed Sequential Processing (HSSP) Image Copy

Although HSSP is not strictly speaking a utility, It has an image copy option available. Further, this option has been significantly enhanced in IMS Version 5. Further information can be found in 3.3.4, “Enhanced Concurrent Image Copy (DFSUICP0)” on page 28.

2.5.4 DEDB Initialization Utility (DFSUMIN0)

This batch utility initializes (creates and formats) a DEDB area which may then be loaded by a user-written BMP or by online processing.

If the area being initialized is registered with DBRC, then its status must be unavailable with recovery needed. These flags are the default settings when the area is registered.

If MADS are to be used, the user may initialize one or more of the ADSs in a single run of this utility. At the completion of this utility, the recovery-needed flag is turned off for each area initialized and the available flag is turned on for each ADS initialized.

2.5.5 DEDB Area Data Set Create Utility (DFSUMRI0)

The ADS create utility is used to create one or more additional copies of an area in a multiple area data set (MADS). It can be used to recover a good area data set from among several damaged ADSs; it can also be used to move an area data set while still online.

Details of this utility can be found in 3.3.1, “DEDB ADS Create Utility (DFSUMRI0)” on page 27.

2.5.6 DEDB Area Data Set Compare Utility (DFSUMMH0)

The DEDB ADS compare utility is used to compare the contents of two or more ADSs for an area. All CIs of each ADS are compared except the first and second, which are not expected to match. The output is:

- A printed dump of up to ten unmatched CIs on SYSPRINT.
- A message showing the total number of unmatched and matched CIs.
- An error queue element status report for each ADS.

2.5.7 Online Area Reorganization

Online area reorganization has always been a unique feature of DEDB. The original utility was called *DEDB Direct Reorganization Utility (DFSUMDR0)* and is described in 3.3.2, “DEDB Direct Reorganization Utility (DFSUMDR0)” on page 27.

IMS Version 5 introduced a replacement utility, called the *High Speed DEDB Reorganization Utility (DBFUHDR0).* While performing the same function, the new utility runs several times faster than the original, and is described in 3.3.3, “High Speed DEDB Reorganization Utility (DBFUHDR0)” on page 27.

Chapter 3. Achieving High Availability and Continuous Operation

One of the most valuable benefits of Fast Path is high availability. From its inception, Fast Path has been designed to provide the most availability possible. It has implemented several attractive features that contribute to increased availability and continuous operation. The availability-enhancing features of Fast Path are summarized under the headings

- DEDB
- Utilities.

3.1 Data Entry Data Base (DEDB)

A DEDB is an availability oriented database. The aspects of a DEDB that contribute to availability are these:

- Partitioning into areas
- MADS
- DEDB record deactivation
- Increasing the size of an area (expansion of IOVF)
- Database control (DBCTL) to give CICS users access to the DEDB availability benefits.

3.1.1 Partitioning into Areas

In a traditional IMS database, a large database implies a large data set or, if multiple data sets are used, the data structure is broken up into segments. But with a DEDB, each data set contains the entire data structure (database record) for a set of roots and all dependent segments of these roots, as shown in Figure 2 on page 20. The DEDB is partitioned into multiple areas. Each area can be represented by one or more ADSs. An area represents a range of DEDB records that are physically kept in an ADS implemented as a VSAM ESDS data set.

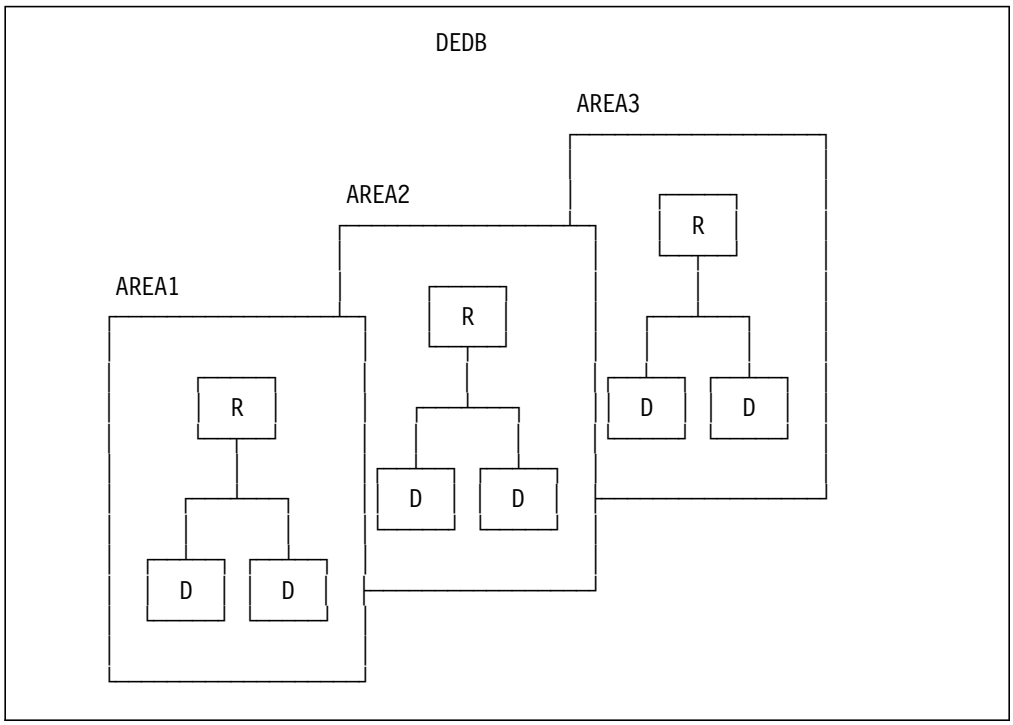


Figure 2. DEDB Area Concept

The area concept offers these characteristics:

- The division of the database into areas is transparent to the application program.
- Large databases can be implemented, because a maximum of 240 areas per DEDB is allowed.
- Areas have a useful degree of independence. Not all areas need to be online and the master terminal operator can start or stop an area. If an area has to be taken offline for any reason, the outage extends only to those records that are in the stopped area. The data contained in the other areas are still accessible.
- Each area can have its own space management parameters. The user chooses these parameters according to the amount of insert activity, which may vary from area to area.
- Control interval sizes up to 28 KB are allowed, in 4 KB multiples.
- ADSs can be allocated on different volume types.
- All maintenance and recovery operations are on an area basis.

3.1.2 Multiple Area Data Sets

Any areas of a DEDB can be replicated, and up to seven copies (ADSs) of an area are allowed. At least one ADS per area is required. The main objective is increased data availability. The multiple ADSs of an area should be on different volumes and can be on different device types.

Write operations are performed to each ADS, but reads are satisfied from one ADS only. On a read error, an attempt is made to read the data from another ADS. A read error is thus completely transparent to the application program. Only in the rare instance where read operations to all the ADSs are unsuccessful

is an AO status code returned to the application program. The system stops an ADS only after a severe error, that is, after a read or write error on the second CI or if more than ten write errors on the data set have been recorded.

The area is stopped only if all ADSs are stopped. Multiple ADSs of the same area can be compared and additional ADSs can be built using online utilities. These utilities are the DEDB area data set create utility and DEDB area data set compare utility. The DEDB area data set create utility also offers an alternative to off-line DB recovery since an error-free ADS can be built from multiple input ADSs, each of which may contain errors, but all of which are on different CIs.

3.1.3 DEDB Record Deactivation

With a partitioned data base like DEDB, the maximum impact of a permanent I/O error is felt when an area must be taken offline for recovery. The scheduling of application programs is not affected as an FH status code is returned for each attempt to access an unavailable area.

Write Errors: Since IMS/VS 1.3, a write error does not result in the area being stopped. Instead, the CI in error is said to be *deactivated*. That is, the identification of the CI in error is remembered, and further retrieve calls for the deactivated CI result in an implicit read error (AO status code). This implementation is referred to as record deactivation and uses an error queue element (EQE) to keep track of each deactivated CI. As the area is no longer stopped, a write error does not affect the entire area but only the affected CI. The data that should have been written out was logged and is therefore fully recoverable. But until a recovery is performed, the data is not available to online processing.

Record deactivation is based on error queue elements. EQEs are used to keep track of the relative byte address (RBA) of CIs with permanent write errors. These EQEs are maintained in main storage but are also written to the second CI of the ADS in error. This technique allows EQE information to be available across all restarts, including cold starts. The recovery operation can be deferred until the area can be more conveniently removed from the system. The only time when an area is stopped for a write error is for what is called a *severe error* situation. This is described as:

- A write error to the second CI or
- More than ten permanent write errors in an ADS

The number of EQEs available can be displayed with a /DISPLAY AREA command.

Read Errors: In general, read errors do not cause an area to be stopped. An AO status code is returned to the applications program call and processing continues. Read errors do not force the recovery of an area unless it is a severe error situation. This would be the case if the area′s second CI could not be read. The second CI contains very critical information, and if it cannot be read, the area is stopped to force a recovery.

Since IMS Version 3, read errors on four different CIs in a MADS environment are treated as a severe error and the affected ADS is stopped. To track these errors, IMS builds a read error queue element (REQE) for each I/O error. These REQEs are maintained in main storage and are not written to the second CI of

the ADS in error. An ADS is stopped for a read error only in a severe error situation. This is described as being:

- A read error on the area's second CI or
- More than three read errors at different CIs of an ADS.

The number of write EQEs available can be displayed with a /DISPLAY AREA command. The full capability of record deactivation is achieved in a multiple area data set (MADS) environment.

3.1.4 Increasing the Size of an Area (Expansion of IOVF)

Before this enhancement became available, if a DEDB area ran out of space, an IMS outage was required to increase the size of that area. This resulted from a data integrity check that IMS performed at area open time. Since IMS Version 3, IMS allows the size of the independent overflow (IOVF) part of an area to be increased without having to stop IMS.

The operational scenario for this is:

- Execute a DBDGEN with the increased root and overflow values
- Execute an ACBGEN.
- Make the DEDB area unavailable by means of a /DBRECOVERY or /STOP AREA command
- Take an image copy the area (in case of fallback).
- Unload the area, using the old DBD.
- Reload the area, using the new DBD.
- Take an image copy the area (to establish the new recovery point).
- Make the area available for processing again, using /START AREA command.
- The new ACBLIB must be used with the next warm or cold start.

3.1.5 Providing DEDB Availability to CICS with DBCTL

DBCTL is an IMS/ESA environment that allows CICS transactions to access DL/I full-function databases and DEDBs. CICS/ESA Version 3 or a later release is required. The DBCTL environment comprises three address spaces, as illustrated in Figure 3.

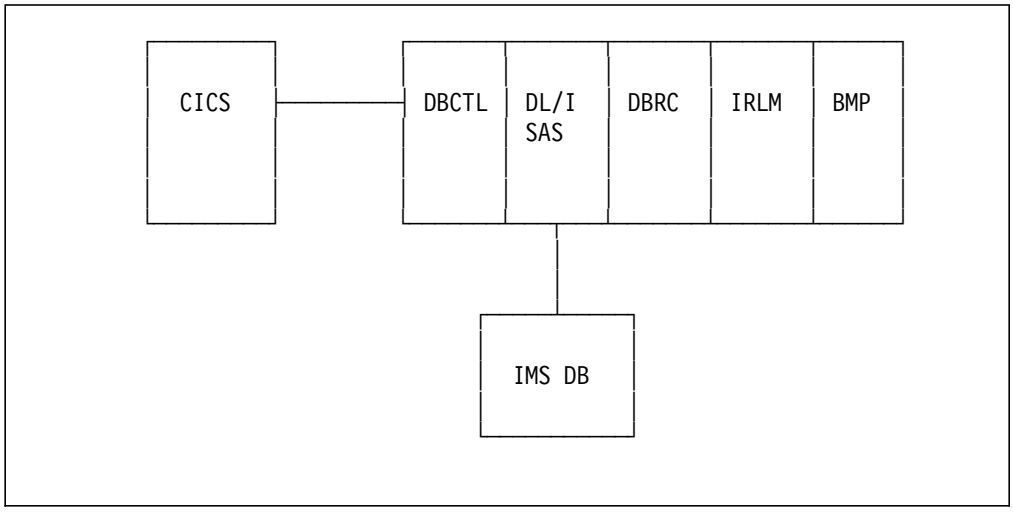


Figure 3. DBCTL Environment

- DBCTL address space
- DL/I separate address space
- DBRC address space

IRLM occupies a fourth address space if it is used.

DBCTL also supports non-message-driven BMPs. BMP regions can access DBCTL databases concurrently with transaction management subsystems, such as CICS.

The coordinator controller (CCTL) consists of the database resource adapter (DRA) and a transaction management subsystem, such as CICS. The DRA allows communication between the DBCTL environment and the attaching transaction management subsystem. One DBCTL environment can service multiple CCTLs, although one CCTL can connect to only one DBCTL. An IMS DB/DC environment cannot connect to DBCTL; It can, however, provide DBCTL services to a CCTL subsystem. In other words, a CCTL can attach to either a DBCTL or a DB/DC system.

In summary, some of the benefits to CICS of DBCTL are:

- CICS applications can exploit the high-availability characteristics of DEDBs.
- DBCTL allows IMS and CICS to share IMS databases more simply.

3.1.6 DEDB Commands (/STOP, /DBRECOVERY AREA....)

There are some differences between the Fast Path and the Full-Function facilities of IMS which the master terminal operator must be aware of:

- /STOP DATABASE command

  In Fast Path, the /STOP DATABASE command when directed at a DEDB prevents the scheduling of new programs sensitive to the database. It allows current programs to continue their processing until they terminate. For DL/I databases, the programs are still scheduled but a call against the database results in either a 3303 pseudoabend, or a BA status code, if the INIT call was issued.

- /DBRECOVERY DATABASE command

  With Fast Path, the /DBRECOVERY command prevents further scheduling of DEDBs. The command to DEDB is always accepted, even if the target database is processed by a BMP region. As a result,

  - The DEDB is stopped.
  - All areas are closed and stopped. FH status codes are returned to further database calls.
  - If eligible, ADSs are dynamically deallocated.
  - A simple checkpoint is taken.
  - An OLDS switch occurs if the NOFEOV parameter is not specified.

  For DL/I databases, the programs are still scheduled, but a call against the database results in either a 3303 pseudoabend, or a BA status code if the INIT call was issued. If a DL/I database is being used by a BMP region, an error message is returned to the master terminal, the command is ignored for the database named in the message, and processing continues for the other databases specified in the command. The master terminal operator

must wait until the BMP concludes processing before reentering the /DBRECOVERY command to close these databases.

- /STOP AREA command

  In Fast Path, the /STOP AREA command prevents access to an area. As a result,

  – The area is stopped and closed.

  – If eligible, ADSs are dynamically deallocated.

  Scheduling of DEDBs is unaffected by this command. If the system processes a /STOP AREA command during HSSP processing, the area is released after the current commit processing completes.

- /DBRECOVERY AREA command

  In Fast Path, the /DBRECOVERY AREA command is equivalent to the /STOP AREA command with the exception that it supports a NOFEOV parameter. In addition, a simple checkpoint is taken. This command is to be used whenever there is a need for removing a specific area from the system, for an offline recovery operation, for example. In this case, the NOFEOV parameter would not be specified and an OLDS switch would take place.

- /STOP ADS command

  With Fast Path, the /STOP ADS command can be used to permanently remove an ADS from processing. The command is rejected if the ADS is the last one for the area. Results of a /STOP ADS command vary with the ACCESS intent of the subsystem:

  – If ACCESS=UP or EX

    - ADS is stopped, closed and deallocated.

    - ADS is marked as unavailable (DBDS record).

    - /STOP ADS acts as a global command with notification to other sharing IMS using IRLM.

  – If ACCESS=RO or RD

    - ADS is stopped, closed, and deallocated.

    - No DBRC interaction occurs.

    - /STOP ADS acts as a local command.

  Stopping an ADS should be done only for a very exceptional reason. Once an ADS is stopped, it should never be brought back to online processing unless you know that no updating took place while the ADS was stopped. This rule is enforced at area open time by the ADS consistency check. The /STOP ADS command should be used very carefully, as illustrated by the following example.

  AREA1 has two ADSs (ADS11 and ADS12). ADS12 has one EQE. It is decided to remove ADS12 from online processing. However, during online update, the operator stops ADS11 instead of ADS12. No fall-back is possible. ADS11 cannot be brought back to online processing and a multiple ADS environment cannot be rebuilt from ADS12 (using the create utility) because it contains an EQE. Offline recovery needs to be performed. In the meantime, online processing could continue with ADS12.

3.2 Main Storage Databases

A MSDB has some special considerations related to start and restart operations, as discussed in the following subsections.

3.2.1 Restart Considerations for MSDBs

Data integrity and restartability of the system require dumping the MSDBs periodically. An image copy of all MSDBs is initiated at startup time, at every system checkpoint, and after a system shutdown request. The actual writing of the data is done by an asynchronous task in the control region while control information is being checkpointed. MSDB processing is not delayed by the MSDB checkpoint.

As soon as the system checkpoint has completed, normal processing resumes in parallel, with the checkpointing of the MSDBs if necessary. The checkpointed data is written alternately onto two data sets called MSDBCP1 and MSDBCP2 (in an XRF environment, MSDBCP3 and MSDBCP4 are also used). Both data sets are preformatted at cold start time or at any warm start specifying the MSDBLOAD parameter. An additional dump of the MSDBs may be taken using the /DBDUMP command. In this case, the image copy is written to a data set called MSDBDUMP. The MSDB dump operation does not serialize the access to the MSDBs, as the image copy is taken in-flight. If the MSDBs are being updated, it is possible that the image copy does not reflect the current state of the MSDBs at the end of the dump operation. In the IMS startup procedure, it is the user′s responsibility to start the MSDB dump recovery utility to extract one specific MSDB (or all MSDBs) from the dump data set. During warm start operation, the master terminal operator has two options:

- Reinstate the MSDBs as they were dumped during the last system shutdown, or
- Request that a new image copy be loaded from the MSDBINIT data set usually following some maintenance activity to one or more of the MSDBs.

In order to make changes to an MSDB, the following steps are required:

1. Select only the required MSDBs (or all of them) from the MSDB dump or checkpoint data set using the MSDB dump recovery utility; that is, create an MSDBINIT data set.
2. Apply changes to the MSDB using the MSDB maintenance utility.
3. Perform a warm start of the IMS system, requesting MSDBs to be loaded from the MSDBINIT data set (with the MSDBLOAD keyword of the /NRESTART command).

As part of the emergency restart process, the MSDBs are recovered using the information from two sources:

- One of the MSDB checkpoint data sets (MSDBCP1 or MSDBCP2)
- The log data set

All MSDB updates from the checkpoint corresponding to the selected MSDB checkpoint data set are applied using the system log.

3.2.2 User Considerations

- Any time the MSDBs are loaded from the image copy data sets, no modifications to the MSDB startup procedure (member DBFMSDBx) are allowed.
- Any modification to the MSDB startup procedure, such as changing the page-fixing indicator or the amount of main storage for dynamic MSDB, requires an initial load operation — that is, a cold start or a warm start with the MSDBLOAD option.

3.2.3 MSDBABEND Option

If the MSDBs are so critical to your Fast Path applications that IMS should not run without them, place a first card image at the beginning of the DBFMSDBx member in your IMS PROCLIB. For each card image, the characters *MSDBABEND=n* must be typed without blanks, and all characters must be within 1 and 72 of the card image. Five possible card images exist, and each contains one of the following sets of characters:

- MSDBABEND=Y

  This card image causes the IMS control region to abend if an error occurs while loading the MSDBs during system initialization. Errors include:

  – Open failure on the MSDBINIT data set

  – Error in the MSDB definition

  – I/O error on the MSDBINIT data set

- MSDBABEND=C

  This card image causes the IMS control region to abend if an error occurs while writing the MSDBs to the MSDBXPn data set in the initial checkpoint after IMS startup.

- MSDBABEND=I

  This card image causes the IMS control region to abend if an error occurs during the initial load of the MSDBs from the MSDBINIT data set, making one or more of the MSDBs unusable. These errors include data errors in the MSDBINIT data set, no segments in the MSDBINIT data set for a defined MSDB, and those errors described under MSDBABEND=Y.

- MSDBABEND=A

  This card image causes the IMS control region to abend if an error occurs during the writing of the MSDBs to the MSDBXPn data set (described in MSDBABEND=C), or during the initial load of the MSDBs from the MSDBINIT data set (described in MSDBABEND=I).

- MSDBABEND=B

  This card image causes the IMS control region to abend if an error occurs during the writing of the MSDBs to the MSDBXPn data set (described in MSDBABEND=C), or during the loading of the MSDBs in system initialization (described in MSDBABEND=Y).

3.3 Utility Enhancements

Some utilities have been enhanced to increase DEDB availability and provide for continuous operation, namely:

- ADS create (DFSUMRI0)
- High speed reorganization (DBFUHDR0)
- Concurrent image copy (DFSUICP0)

3.3.1 DEDB ADS Create Utility (DFSUMRI0)

The ADS create utility is used to create a second or subsequent copy of an existing area in a MADS environment. Phase 1 of this utility initializes the new ADS. It is not necessary to use the DEDB initialization utility before running the create utility. During this phase, online updates are made to the existing ADS only. Phase 2 copies data from the existing ADS to the newly initialized ADS. During this phase, online updates are made to all ADSs.

A useful feature that aids availability is the ability to move an ADS from one disk to another. An extra copy of the area is defined in DBRC, and then the create utility is used to build a new ADS on the target volume. When this completes, the original copy can be stopped and deleted.

3.3.2 DEDB Direct Reorganization Utility (DFSUMDR0)

This utility exists up to IMS Version 4. It is used to reorganize an area while it remains online to other transaction and BMP processing. It exploits a reserved unit of work called the *reorganization UOW*, which is used as a work area. Reorganization is performed for each UOW in two phases:

BUILD phase: During the BUILD phase, segments are read in hierarchical order from one data UOW and written into the reorganization UOW. A read or write error during this phase causes the utility to terminate. However, since the data UOW has not yet been modified, the area remains usable and available. At the completion of this phase, the reorganization UOW contains the reorganized data.

COPY phase: During this phase, the control intervals in the reorganization UOW are written back to the original data UOW. A read error during this phase stops the area and sets the recovery-needed flag set on (assuming MADS is not used). A write error results in the building of an EQE for the ADS suffering the error. Unless the error is severe, the utility continues and the ADS remains usable.

3.3.3 High Speed DEDB Reorganization Utility (DBFUHDR0)

The high-speed DEDB reorganization utility that became available in IMS Version 5 replaces DFSUMRI0. The new reorganization utility, like the old one, works on a single UOW at a time, but does not use the reorganization UOW. The UOW is reorganized in main storage and written back to the original UOW as a single unit of recovery. The reorganization utility uses the standard resource management of IMS Fast Path. Each UOW being reorganized is held exclusively by the reorganization utility until this UOW has been reorganized, and has been written back to its permanent location. The UOW cannot be accessed by other programs during this period. The standard IMS Fast Path commit process is

used to write only those CIs that are changed. This use of storage gives improved performance and reduces logging.

For the high-speed DEDB reorganization utility, the input parameter BUFNO specifies the number of buffer sets, each large enough to hold a full UOW, rather than the number of buffers. For example, a UOW of 16 CIs would require a buffer set of 16 buffers, each buffer holding one CI. Buffers are obtained dynamically from a private buffer pool which is created when the utility begins processing. This buffer pool can be extended as necessary, up to twice the original size. A default value of three buffers sets (allowing up to six) is used if no value is specified on the BUFNO parameter. This default of three allows one buffer set for the UOW being reorganized, one buffer set for the reorganized output, and one buffer set for any overflow access. If waits for buffers are experienced, the buffer pool is extended until the maximum is reached. Thus, the utility maximizes performance while using the minimum number of buffer sets. The buffer pool is permanently page-fixed in real storage for performance reasons. If real storage is limited, then the page-fixing of the buffer pool affects how many copies of the reorganization utility you run simultaneously. For each area that is being reorganized, a private buffer pool is created and page-fixed. The amount of real storage used can be significant. For example, an area with 48 CIs, each CI being 16 KB in size, using the default BUFNO specification, would require 2.3 MB of real storage before any buffer pool extensions.

If storage is not an issue, even better performance can be achieved by requesting a BUFNO value of four. In this case, look-ahead buffering is done using the additional buffer set.

3.3.4 Enhanced Concurrent Image Copy (DFSUICP0)

IMS Version 4 contains a continuous operations enhancement for Fast Path users of the concurrent image copy (CIC) utility. The CIC allows updates by application programs at the same time the utility is producing a copy of the data. CIC is thus an important continuous-operation feature of IMS. Although it is online in the sense of allowing concurrent update during the copy, it is actually a variation of the batch image copy utility and, as such, does not have access to the online buffers and control blocks.

Before IMS Version 4, it was recommended that an area be stopped and started again just before taking a CIC. This minimized the amount of log data that would have to be input to DB recovery should recovery be required. Clearly, this interruption was undesirable, especially when aiming for continuous operations. With an IMS Version 4 enhancement, this interruption is no longer necessary, and continuous availability can be maintained while the CIC utility is run.

3.3.5 HSSP Image Copy Option

The high-speed sequential processing image copy option is provided to allow the user to request that a single or dual image copy be taken at the same time that the entire DEDB area is being sequentially read, updated, or both. This option reduces elapsed time significantly for users who follow sequential updates of areas with image copies, since both the copy and the update processing can be done in one pass. Before IMS Version 5, the use of the HSSP IC option also reduced logging, since not all the application updates were logged (they were captured on the image copy). However, in some cases this created operational complexity (for example if the program abended before completing the image

copy). Consequently, in IMS Version 5, the image copy option has been rewritten to be simpler to use and to run even faster.

3.4 Fast Path Log Timer

To maintain adequate response time in low activity situations, the output thread processing and the sending of Fast Path output messages can be triggered by explicit log write ahead requests based on a time interval. Before IMS Version 1.3, the time interval could vary depending on the rate at which the logging data were generated. In rare instances where the update activity was low but constant, transaction response time was occasionally large. In IMS Version 1.3, the time interval had a fixed value of one second. In IMS Version 3, the time interval was reduced to 0.35 second and the above problem was eliminated.

Chapter 4. Achieving High Performance

As its name suggests, Fast Path provides the very high performance it was designed for. This chapter describes some of the features that contribute to its excellent performance. Also considered are some of the newer features that have made Fast Path even faster. The information is presented under the following topics:

- Data entry database (DEDB)
- Main storage database (MSDB)
- Fast Path buffers
- Expedited message handler (EMH)

4.1 Data Entry Data Base (DEDB)

The performance-related aspects of DEDB are these:

- Path length
- I/O elimination and parallelism
- Sequential dependent segments (SDEPs)
- High-speed sequential processing(HSSP)
- Subset pointers
- Virtual Storage option (VSO)
- Log reduction

4.1.1 Path Length

DL/1 calls against DEDBs use significantly less CPU than the same calls against a full-function database. This efficiency reflects the fact that Fast Path is a subset of full function but it does some things in a different way.

Because DEDB functions are a subset of the full DL/1 set, Fast Path does not have to test for all the many options, (such as secondary indexes, logical relationships, backward pointers, fixed or variable length, or which subpool to use). This in itself produces a measurable benefit.

DEDB updates are not logged at call time. Instead, Fast Path simply notes what data was changed. Then, at sync point, the noted information is used to build the log records, and the IMS logger is called. Thus, the logger is called only once per transaction, instead of at every update call.

DEDBs do not use buffer lookaside. Because the databases are expected to be large, it is assumed that the data required by a DL/1 call will not already be in the buffer pool. So, unlike with full-function, the buffer is not searched.

The DEDB locking strategy also helps to reduce CPU path length. Locks are generally taken at the CI level, so fewer locks are needed than with full-function databases. There are only two levels of lock, and they are usually held until sync point, when appropriate ones are all released together. Again, this strategy is much simpler than that used by the full-function IMS.

DEDB data sets are VSAM entry-sequenced data sets (ESDSs). Even so, Fast Path uses media manager to access them, rather than the standard VSAM macros such as, GET and PUT. This reduces CPU cost, as does the fact that the output threads run under Service request blocks (SRBs) rather than time-controlled blocks (TCBs).

In summary, one would typically expect a DEDB call to use about half the CPU path length of the same call against a hierarchical data access method (HDAM) database.

4.1.2 I/O Performance

4.1.2.1 Input/Output Elimination

Fast Path uses several techniques to reduce or eliminate I/Os. Root synonym chains from one CI to another are followed only when it is clear the target root segment is not present in the CI already accessed. SDEPs are written out only when a buffer is full. Only afterimages of DEDB updates are logged, thus saving log I/Os.

IMS Version 5, eliminates DEDB I/O reads (apart from the first time each CI is read after an area is opened) as the data is held in memory. Although not all I/Os can be eliminated, techniques are available for improving or reducing the impact of the I/Os that remain.

4.1.2.2 Output Threads

The objective of the OTHREADs is to perform DEDB updates asynchronously from transaction processing. Different area data sets are updated in parallel with each other and concurrently with transaction read processing. In general, because it is asynchronous, I/O write performance is not a potential bottleneck. However, it is still important to optimize write performance so that device usage is minimized to avoid DASD contention, and so that I/O-bound BMPs achieve their optimal performance.

The maximum number of concurrent output threads is specified at system definition time (OTHREAD parameter on FPCTRL macro) and can be overridden at startup time (OTHR parameter). Each output thread defined provides for the scheduling of a separate SRB to control the writing of the modified DEDB CIs. In releases prior to IMS/VS 1.3, all CIs updated during one sync point interval were passed to one output thread. Since IMS/VS 1.3, the results of a sync point interval (updated DEDB CIs) are likely to be written by multiple output threads. The system attempts to schedule as many output threads as there are ADSs involved. For example, if an area has three ADSs, the system will attempt to schedule up to three concurrently active output threads, provided there are at least three CIs to write back. A too low output thread value could result in buffers waiting for an output thread. This, in turn, could lead to CI contention as CIs are held exclusively until they have been written out.

In IMS Version 4, chained writes are introduced, and the output thread processing is changed to be even more efficient in output thread scheduling. Consequently, IMS can update DEDBs more efficiently and release the CI lock sooner, which leads to even better performance. The new output thread processing has the following characteristics:

- One OTHREAD handles all writes for a given area (with or without MADS):
  - Candidate CIs are sorted in relative byte address (RBA) sequence.
  - Chained writes are used
- With MADS, a single OTHREAD writes all CIs to one ADS and then writes the same CIs to the next ADS.
- The maximum number of output threads is determined by the value in the OTHREAD parameter. If this is not specified, then the default value is 25% of the total number of areas, up to a maximum of 255 output threads.

4.1.2.3 DASD I/O Contention

I/O contention, on direct access storage devices (DASD) can, as always, can limit DEDB performance. To minimize this impact:

- Allocate heavily used areas and system data sets, such as, OLDS, on different DASD volumes and channels.
- Limit the number of application programs accessing any one DEDB area.

  One possibility here is to design the transaction, Fast Path input edit and routing exit, and randomizing algorithm combination so that access to any one area is limited to a particular application program or programs.
- Limit the incidence of buffer stealing, or its impact, by appropriate design of the application programs. Buffer stealing may force a second I/O to recover the stolen buffer or CI if the logic of the application program requires a CI for a significant number of DL/I calls after it was first retrieved.

4.1.2.4 I/O Dispatching Priority

DEDB writes are performed by the OTHREADs running under SRBs in the IMS Control Region. By default, then the write I/Os, have a higher priority than the reads, which execute under dependent region TCBs. This is probably not what is required. Synchronous tasks, such as the reads, normally want higher priority than asynchronous tasks (the writes), especially since most applications read more CIs than they update. Should an application become I/O bound, there will simply be a shortage of buffers and reads will not be issued until the writes have made space in the pool.

We recommend that you set the I/O priority for DEDB processing regions higher than that for the control region.

4.1.2.5 Areas and MADS

Being able to partition a DEDB into areas provides the opportunity to control the I/O rates to the data sets. In other words, spreading the database over several areas allows the I/Os to be distributed over multiple volumes.

MADS also can provide performance benefits. The read I/Os to the area are spread across all the available ADSs, reducing the I/O load on any one volume.

4.1.3 Sequential Dependent Segments (SDEP)

Sequential dependent segments (SDEP) are stored separately from their roots in an entry-sequenced manner at the end of the area. SDEPs are chained from their roots in a last-in first-out sequence. The SDEP facility is designed for data entry processing with a fast insert capability. Typically, the SDEP segment is used for data collection, journaling, auditing, and the like, where an application is likely to deal with large amounts of data to be processed offline later. SDEP

segments are typically retrieved with the SDEP sequential dependent scan utility and then processed in a batch environment. The SDEP scan utility runs online and does not require access to roots.

Once inserted, SDEP segments can never be updated. Delete can only be done using the DEDB sequential dependent delete utility (mass delete only).

Online retrieval using DL/1 get calls should be kept to a minimum because the access is likely to be inefficient. A physical input operation is generally needed for each segment.

4.1.4 High-Speed Sequential Processing (HSSP)

High-speed sequential processing (HSSP) is designed to benefit those programs that do large volumes of physical sequential processing against DEDBs. HSSP can be beneficial for read-only as well as update jobs.

In release before IMS Version 5, the key performance benefit is derived by taking advantage of the 3990-3 Extended Functions (assuming the DEDB resides on a volume controlled by this or a more recent cached DASD controller). HSSP uses the sequential access mode of the controller, which allows tracks to be read before their CIs are requested by the host. With the 3990 cache controller, IMS keeps approximately five tracks worth of data in the cache at any one time. If the BMP is truly sequential (that is, the majority of CIs in the unit of work are being accessed), the data being processed should be found in the cache most of time, thus removing the need to do a DASD I/O to read the data. HSSP also takes advantage of the DASD fast-write capability of the 3990-3 Extended Function. The net effect should be to reduce the elapsed time of a BMP job.

Also contributing to the enhanced BMP performance is the use by HSSP of private buffers, rather than the common Fast Path pool; this reduces buffer contention. A further contributor is the HSSP locking mechanism. HSSP locks on UOWs, not on CIs, which results in fewer lock requests.

IMS Version 5 introduces a completely rewritten HSSP that uses software techniques and main memory, rather than relying on DASD hardware, to provide asynchronous read-ahead processing. This results in even higher and consistent levels of performance.

4.1.5 Subset Pointer (SSP)

A subset pointer points from a parent to a child segment. The target segment can be any segment in the child′s twin chain. A subset pointer creates a subset of the twin chain — namely, all the segments from the one pointed at to the end of the chain. The objective is to enable applications to access the subset without having to process down the twin chain from the beginning. Some applications commonly access only the last few segments in a long twin chain. In those cases, the benefits of SSPs can be quite significant, saving I/Os as well as reducing the CPU cost.

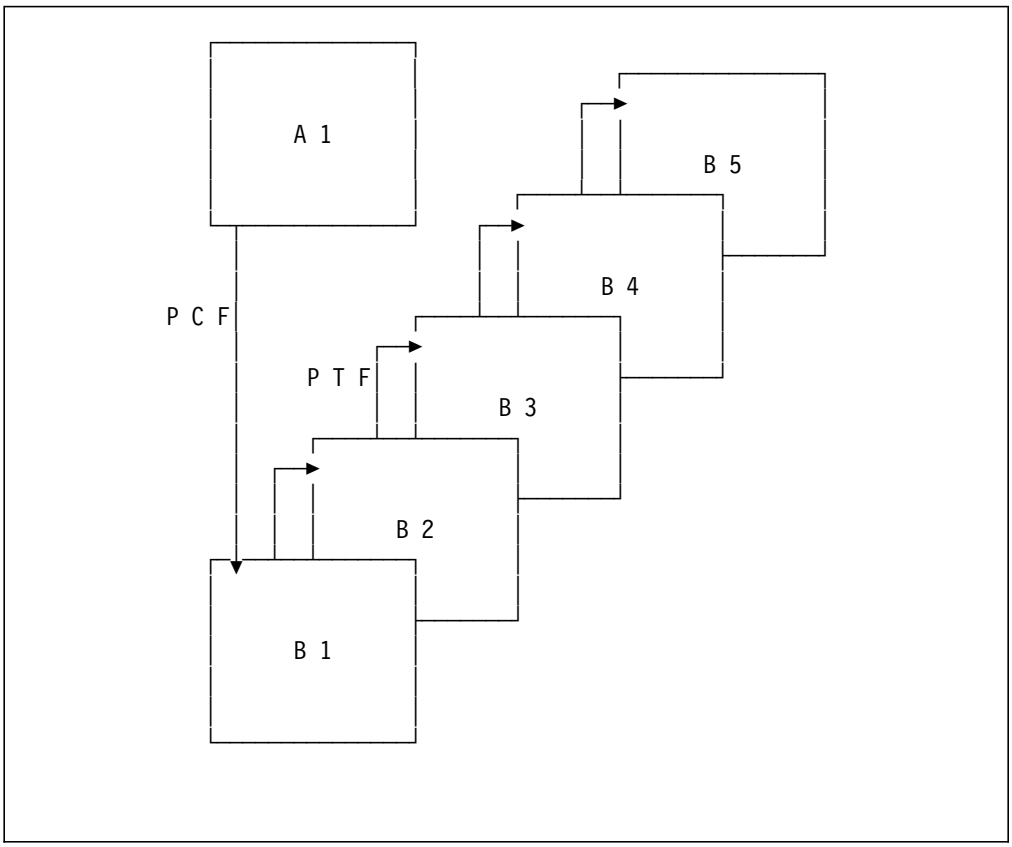


Figure 4. Segment Chain without SSP

In Figure 4, the application program (or IMS if an appropriate segment search argument is specified) has to retrieve segments B1, B2, and B3 before accessing segments B4 and B5.

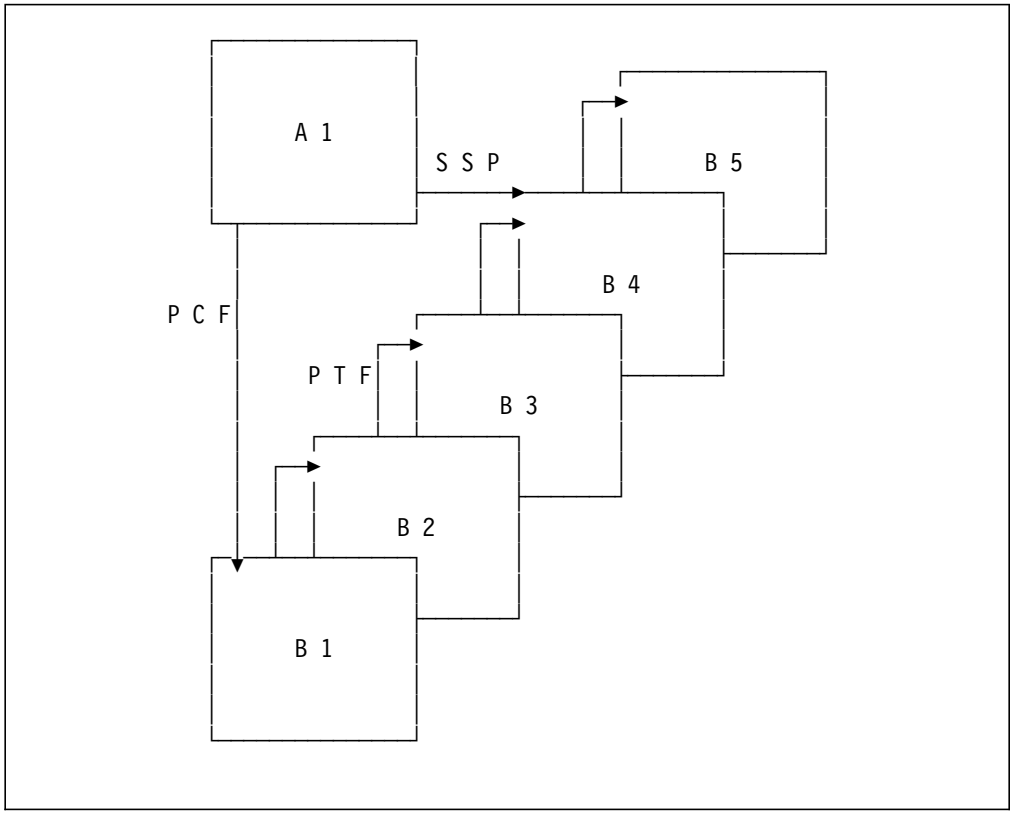


Figure 5. Segment Chain with SSP

Figure 5 shows an example of the same data base with one subset pointer. We assume that the pointer was set by a previous call to point to segment B4. Using the subset pointer, two calls are required to retrieve segments B4 and B5 instead of five calls in the previous example.

An example of the use of the SSP is given by a banking application in which segment type A is an account segment and segment type B is a money transaction segment. If the money transaction appears on the owner's passbook, it is said to be posted. This assumes the passbook was present at the time the transaction took place. On the other hand, some money transactions might not be reflected in the passbook. In this application, the SSP indicates the first money transaction not yet posted to the owner's passbook. All preceding segments have already been posted to the passbook.

Subset pointers can only be used with DDEP segments. A maximum of eight SSPs can be defined for each DDEP segment type. All SSPs are independent, so multiple SSPs can point at the same segment occurrence.

Information on how applications use and manipulate subset pointers can be found in 5.2.6, "Using Subset Pointers" on page 56.

4.1.6 Virtual Storage Option (VSO)

Beginning with IMS/ESA 5, the Fast-Path virtual storage option allows you to map some or all areas into virtual storage (into an MVS data space). For high-end performance applications with DEDBs, using VSO allows you to realize the following performance improvements:

- Reduced read I/O — Once a VSAM control interval (CI) has been brought into virtual storage, all subsequent I/O read requests read the data from virtual storage rather than from the DASD.
- Decreased locking contention — For VSO DEDBs, update locks are released at sync point, once the updates have been copied into the data space. (For non-VSO DEDBs, locks are held until after the updated CIs have been written to DASD.)
- Fewer and more efficient writes to the area data set — Updated CIs in memory are not immediately written to DASD; instead they are kept in the data space and written to DASD at system checkpoint or at regular intervals (1 or 4 minutes, depending on the area size). To optimize these writes, CIs are first sorted into relative-byte-address order, and written with chained I/Os.

In all other respects, VSO DEDBs are the same as non-VSO DEDBs. Therefore, VSO DEDB areas are available for IMS DBCTL and LU6.2 applications, as well as other IMS transaction manager applications. Use DBRC commands to specify that you want to use a DEDB as a VSO DEDB.

Customers are expected to convert their MSDBs to VSO DEDBs. This process is made feasible by providing DEDBs with functions that were previously unique to MSDBs, namely:

- The field (FLD) call
- Fixed length segments
- The MSDB or DEDB commit view

In addition, VSO DEDBs have the following:

- Full DBRC support
- Full two-phase commit sync-point processing

  Consequently, unlike MSDBs, VSO DEDB areas are available to DBCTL and LU6.2 applications.
- HSSP support for VSO DEDBs
- Availability of DEDB utilities
- Online database maintenance
- Support for the full hierarchical model

  Insert and Delete calls, not possible with MSDBs, are now available with VSO DEDBs.
- Improved search techniques — MSDBs use

  the binary search technique; VSO DEDBs use the randomizer search technique.
- MSDB checkpoint data sets are not required for VSO DEDBs

VSO DEDBs have the following conditions for their use:

- VSO areas must be registered with DBRC.
- Block-level data sharing is not allowed while the area is in virtual storage (with IMS Version 5)

That is, you cannot register a VSO DEDB with DBRC if its SHARELEVEL is greater than 1.

- The maximum allowable size for a VSO DEDB is 2 GB. You can use the /DISPLAY FPVIRTUAL command to determine how much actual storage a particular area has.

4.1.7 Log Reduction

Before IMS Version 3 (that is, with IMS/VS), the image of an entire segment was logged when it was replaced, even through much of the data might have remained unchanged. Since IMS Version 3 (that is, with IMS/ESA), IMS typically logs only the changed parts of a segment. Figure 6 on page 39 illustrates the updates made to a DEDB segment, and the log data that would be produced on an IMS/VS system compared with an IMS/ESA system. In some cases, IMS/ESA will log the unchanged parts of a segment as well, if it results in less total log data. Each log record has a header of 104 bytes. So if two sets of updated data are within 104 bytes of each other, it is more efficient to log both changes (together with the unchanged piece in the middle) in a single log record.

For each CI update, IMS keeps track of where in the buffer the change starts, and how long the changed data is. IMS uses the Fast-Path buffer prefix to track these updates, and multiple slots are provided to store the location and length information. At sync point time, these slots are read to build the log records. When a CI update is within 104 bytes of a previous update, the previous slot is also updated to incorporate the new update.

The number of slots in the buffer prefix is specified by the LGNR IMS execution parameter. The safest value is calculated by taking the CI size and dividing it by 104. This resulting number of slots is guaranteed to be sufficient to control any number of updates to the CI. If a smaller number is specified, to save buffer prefix storage, IMS could run out of slots to save the update location information. If that happens, IMS recalculates all the slot values based on a combining constant of 208, rather than 104. If this is still not enough, IMS works with 416. Should the number of slots still be too small, IMS logs the whole CI instead of the individual changes.

Both recalculating the slots with a new combining constant and logging whole CIs are inefficient, and to be avoided if possible. The Fast Path Log Analysis Utility (DBFULTA0) reports on how often either event has happened.

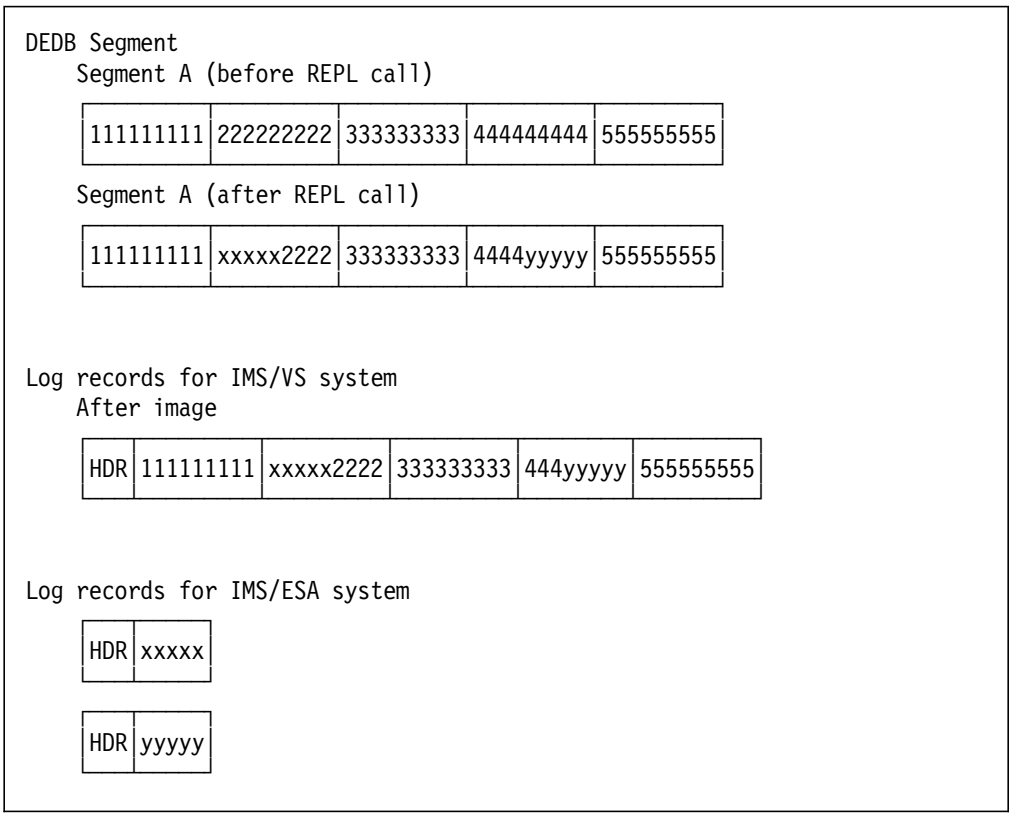


Figure 6. Log data for a DEDB Update

4.1.8 Unit of Work (UOW) Size

The UOW is the unit of space allocation used to define, for an area, how large the root addressable and the independent overflow parts are. Two factors might affect the choice of UOW size:

- DEDB direct reorganization utility runs on a UOW basis. Therefore, while the UOW is being reorganized, none of the CIs and data they contain are available to other processing. A large UOW could cause resource contention, resulting in increased response time if the utility is run during the online period.
- The UOW is also the basic unit of processing when running BMPs with PROCOPT=P. Such BMPs get their GC status at the end of each UOW, and take a SYNC or CHKP at that time. They require enough buffers (as specified in Normal Buffer Allocation (NBA)) to hold the UOW, and maybe more if IOVF CIs are processed as well. The total number of buffers required is determined by the number of parallel BMP streams and the UOW size.

UOW size is otherwise an arbitrary value. Users often set it, for want of any better indicator, as one or two tracks worth of data on the DASD (though in fact there is no link between UOW size and DASD occupancy).

4.1.9 Contention for CIs

DEDB locking is generally done at the CI level. When two application programs concurrently request access to the same CI, and one or both have update access, then one requester is forced to wait. If such a wait results in a deadlock, then one of the application programs is selected for termination. Its current sync interval is pseudo-abended and forced to release its resources. If it is a message-driven program, then the message is immediately rescheduled. For a BMP, processing is resumed from the previous sync point.

The number of CI contentions and deadlocks can be decreased by taking the following steps:

- Ensure that the randomizing routine spreads all root keys as evenly as possible among the available root anchor point (RAP) to minimize the number of roots per CI.
- Transactions or sync intervals with that are intended for update but which read significantly more CIs than they update might benefit from deliberately forcing more frequent buffer stealing. Reducing the NBA size is one way of doing this, so that buffers used for reads can be released more quickly. The DEQ DL/1 call, introduced in IMS Version 5, also allows the explicit invocation of buffer stealing. In either case, when the buffer is released, the exclusive lock on its CI is released.

Reports produced by the Fast Path log analysis utility give statistics about CI contentions.

4.2 Main Storage Data Bases

Main Storage Data Bases have always provided exceptionally high performance. The characteristics of an MSDB are:

- No database I/O
- Root-only database
- Minimal resource contention when FLD call is used.
- Reduced path length
- Only after-images are logged

4.2.1 Field (FLD) Call

The Field (FLD) call is used to update a segment efficiently. It can be used with MSDBs or DEDBs from IMS Version5. Typically, a single FLD call is used to specify a number of tests to be performed on some fields (verifications), and then, if all tests are positive, it specifies how one or more fields are to be updated (changed). The FLD call is more efficient than a GHx and REPL combination of calls for several reasons:

- The FLD call has a shorter path length.
- The segment is locked only at sync point time, and not from call time to sync point time.
- When used on an MSDB segment, the FLD produces significantly less log data and requires less buffer space than a GHx and REPL combination.

In summary, programming with FLD logic can contribute to higher transaction rates and shorter response times.

4.2.2 Virtual Storage Requirement

MSDBs reside in main storage rather than on DASD. Prior to IMS Version 2, the MSDBs were resident in the common storage area. As part of the Virtual Storage Constraint Relief (VSCR) in IMS Version 2, however, the MSDBs were moved above the 16 MB virtual storage line; therefore MSDBs reside in ECSA rather than CSA. This means that MSDBs can be used for databases whose size is measured in megabytes rather than kilobytes.

4.3 Fast Path Buffer Pool

To provide access to the Fast Path databases, there is a single buffer pool called the *global* or *common fast path buffer pool*. Consistent with Fast Path's objective of keeping CPU path lengths as low as possible, this pool is managed as a single pool, with all buffers the same length. Similarly, the buffer management techniques are designed to be simple, but a consequence is that application and systems programmers need to be more aware of how buffers are used than when using, for example, DB2 or DL/1 full-function databases.

4.3.1 Making Buffers Available

The specifications of the pool are given at IMS system definition time and can be overridden at IMS startup time. Three parameters are used to define the Fast Path buffer pool:

- DBBF: Total number of buffers
- DBFX: System buffer allocation
- BSIZE: The size of each of the buffers, which must be larger than or equal to the size of the largest CI of any DEDB to be processed.

The Common Pool is defined to contain a certain number of buffers, and these buffers are allocated in virtual storage at IMS startup time. However, buffers are not usable until they have been page-fixed. The whole pool is not page-fixed. Instead, certain events cause a number of buffers to become fixed, and thus usable:

- A dependent region is given access to Fast Path by including two execution parameters, namely NBA=m and OBA=n, in the region JCL.
- When the first dependent region with access to Fast Path starts, a system-defined number (DBFX) of buffers are immediately page-fixed.
- As each dependent region starts, the NBA value determines how many additional buffers are page-fixed.
- Also, as each region starts, IMS assesses all the current OBA values. If the OBA of the starting region is greater than any previous OBA, then extra buffers are page-fixed up to the new OBA. For example, if the previous largest OBA had been 15 and the new region's OBA is 25, then 10 more buffers will be page-fixed.
- When an area is opened for a DEDB that has an SDEP defined in the hierarchy, then one more buffer is page-fixed.

The set of page-fixed buffers represents the available buffer pool. Buffers are allocated individually to requestors as required. There is no sense in which sets of buffers are allocated to a region long term. Any buffer can be used for any purpose by any requestor.

4.3.2 Use of Fast Path Buffers

Fast Path buffers are used:

- To hold updated MSDB segments
- As a work area for FLD calls (until sync point)
- To temporarily store inserted SDEPs (until sync point)
- To hold DEDB CIs
  - Those currently being processed
  - Those updated by previous processes and waiting to be written back to the database
- As SDEP Collection Buffers. At sync point time, inserted SDEPs are moved here from the temporary buffer. Only when this buffer fills up does it get written out to DASD. There is one such buffer allocated for each open area defined with an SDEP.

4.3.3 Dependent Region Usage of Fast Path Buffers

The number of buffers a transaction or a BMP sync interval is allowed to use must be specified for each region. The NBA and OBA parameters mentioned above are also used for this purpose.

- Normal Buffer Allocation (NBA)

  IFP, MPP, and BMP regions accessing Fast Path resources require this number to be specified in the region startup procedure. NBA should be set so that the vast majority of transactions or BMP sync intervals in that region do not need more than this number of buffers.

- Overflow buffer allocation (OBA):

  This specifies a number of additional buffers that can be made available to the region in exceptional circumstances. The total of NBA+OBA can never be exceeded. Only one region at a time can be using its OBA. This serialization is enforced by the exclusive holding of the OBA latch.

4.3.4 Sizing the Fast Path Buffer Pool and the Available Pool

It is clearly essential to provide enough buffers (DBBF) to fully meet the system requirements at all times. However, there is no benefit in making the available pool larger than necessary, since Fast Path does not exploit buffer lookaside.

The number of buffers allocated by the system at any point in time is given by the following formula:

A = Number of opened areas with SDEP segment type defined

NBA = Normal buffer allocation of each active region

N = Total of all NBAs

OBA = Largest overflow buffer allocation

DBFX = System buffer allocation

DBBF = Fast Path buffer pool size as specified by the user

At any point in time, the following formula must hold true:

DBBF must be greater than or equal to A + N + OBA + DBFX

However, the real requirement is for sufficient page-fixed buffers. Updated DEDB CIs, held in Fast Path buffers, are only written out to DASD after the corresponding log records have been physically written out. These log writes are not forced (other than by the log timer which fires if log records have remained unwritten for more than 0.35 second). Consequently, updated DEDB buffers remain unavailable after sync point time until log I/O and database I/O have completed. The purpose of DBFX is to create page-fixed buffers specifically available for holding these pending updates while the currently executing programs have available their NBAs (and largest OBA).

In reality, at any point in time, it is unlikely that every region will be using its full allocation of buffers. When a program emerges from sync point, it has no buffers. Typically, the maximum it needs, reached shortly before sync point, is less than the NBA. On average, a region uses less than half its NBA at any one time.

A system with a variety of light update and enquiry transactions would probably require only a small DBFX. But a system with many heavy update BMPs would require a large DBFX.

The Fast Path log analysis utility should be run regularly to examine the region′s buffer use and to check for buffer shortages.

4.3.5 DEDB PROCOPT=P

Control intervals in buffers waiting to be written to DASD are exclusively locked. No programs can access them, and this includes the program that has just made the updates and performed sync point processing. For transactions and randomly processing BMPs, this is rarely a problem. But for BMPs that sequentially process through an area (that is, in physical sequential order), it is important that CIs updated before a sync point are not immediately rerequested after the sync point.

To ensure that rerequesting can be avoided, Fast Path provides the BMP processing option, PROCOPT=P.

The PROCOPT=P option is specified during the PCB generation in the PCB statement or in the SENSEG statement for the root segment. The option takes effect only if the region type is a non-message-driven BMP. If specified, it offers the following advantage:

- Whenever an attempt is made to retrieve or insert a DEDB segment that requires a UOW boundary be crossed, a GC status code is set in the PCB and no segment is returned or inserted. The calls for which this applies are: G(H)U, G(H)N, POS, and ISRT. While the crossing of the UOW boundary has

no particular significance for most applications, the GC status code indicates that this is the ideal time to take a sync point when processing sequentially. A database record cannot span two UOWs. A root and all its direct dependents are in CIs belonging to one UOW (which may include IOVF CIs allocated exclusively to that UOW). Consequently a request for a database record in the next UOW cannot possibly require access to CIs in the previous UOW (and which are now locked pending being written out to DASD).

The sync point is invoked by either a SYNC or a CHKP call, which normally causes the position on all currently accessed databases to be lost. The application program would then have to resume processing by first reestablishing the sequential position. This situation is not always easy to solve, particularly for unqualified G(H)N processing. A feature of using PROCOPT=P is that when a SYNC or CHKP call is issued after a GC status code, the database position is kept for the sequential PCB. The database position is such that an unqualified G(H)N call issued after a GC status code would return the first root segment of the next UOW.

For a BMP that uses PROCOPT=P, the NBA should be set to accommodate a complete UOW (including typical requirement for updated overflow CIs), together with the requirements of other Fast Path PCBs. Then add an extra one or two to the NBA to ensure that the GC status occurs before there is a chance of running out of NBA buffers.

In exceptional cases, where processing of a UOW requires more buffers than NBA, the application will receive FW status codes, will use its OBA, and should take a sync point as soon as possible. There will almost inevitably be a delay before processing can resume, waiting for the most recently updated CIs to be written out and made available for rereferencing.

4.4 Expedited Message Handler

The expedited message handler (EMH) provides an alternative to full-function input and output message queuing and application program scheduling. The EMH is a performance-oriented component of Fast Path and has characteristics as follows:

- Handles input and output message processing

  Unlike the full-function approach of using a message queue mechanism, EMH has a dedicated buffer for every terminal that is eligible to enter Fast Path transactions.[3] Each terminal may define a different size buffer, but that size must be at least as large as the largest input or output message that will be received from or sent to the terminal.

- Handles queuing and scheduling of Fast Path transactions

  EMH does not make use of the IMS message queues. Instead, it uses a simpler approach to queuing and scheduling, resulting in shorter path length and providing a performance advantage. When a Fast Path region is started, a specific PSB (and application) is scheduled into that region. Associated with each PSB is a balancing group (BALG), which is the anchor point for the queue of messages in the terminal buffers for the associated application program. EMH uses a simple first-in, first-out (FIFO) approach. The

[3] From IMS Version 4, all terminals are eligible to use EMH.

objective is to schedule the transactions for processing as quickly and efficiently as possible. EMH has the following requirements:

- Requires single-segment input and output
- Requires response mode

In general, the performance advantage of using EMH derives from the fact that a shorter path length is involved with the processing of functions handled by EMH, as opposed to its full-function counterpart. The simplified approach to scheduling mentioned above results in a shorter path length, and therefore contributes to Fast Path′s ability to handle larger volumes of transactions. The expedited method of handling messages and scheduling was geared toward applications with simple transactions. The philosophy is to get transactions in and out as quickly as possible, without requiring any unnecessary frills such as priority scheduling.

4.4.1 EMH with SLUP Devices

Terminals defined as system logical unit type P (SLUP) or FINANCE can specify whether to use Fast Path acknowledgement technique (FPACK) or the standard acknowledgement technique (NFPACK). FPACK is the default and provides the possibility of slightly better network performance.

With NFPACK, when Fast Path sends a transaction reply, it requests the SLUP program in the workstation to acknowledge it with a definite response (DR2). With FPACK, this System Network Architecture (SNA) acknowledgement is not requested. Instead, the next input message to IMS is also treated as an acknowledgement to the previous output.

The potential consequences of using FPACK should be appreciated however. If when a reply is sent, there is another message on the queue for the same SLUP program, then a definite response is requested. Otherwise, an exception response will be requested. If the SLUP program has no more input, the reply remains unacknowledged until the next input is submitted or the SLUP program issues an SNA ready to receive (RTR) command. IMS keeps the output message enqueued in the terminal buffer in case it has to be resent, and does not send any further messages to that destination.

Given the high bandwidth of modern networks, it is generally recommended that NFPACK be specified to eliminate delays in IMS sending output messages.

4.4.2 EMH Buffer Pool

In IMS Version 4, the management of EMH terminal buffers was completely rewritten. The Fast Path terminal buffers, as of this version, are dynamic and reside in a new pool called the EMHB pool. Buffers are acquired when transaction input is received, and are freed when no longer needed.

Chapter 5. When and How to Use Fast Path

The preceding chapters deal with the theory of Fast Path, what it can and cannot do, and some of its restrictions. This chapter deals with the practicalities of using Fast Path. You need to know the strengths and weaknesses of each feature of Fast Path in order to make an informed decision about which to use.

Many of the sections in this chapter refer to functions of our sample application. You might find it useful to refer to the description of this in Chapter 6, “Sample Application” on page 69.

5.1 Factors Affecting Both MSDBs and DEDBs

Most factors applying to MSDBs apply equally to DEDBs.

5.1.1 Field Calls

We met the field call in 1.2.2.3, “Field Calls” on page 6. Its has two important characteristics:

- Locking is only for a very short time at commit
- You can verify that the field does not change between read and commit.

5.1.1.1 Field Calls and Arithmetic

You can use a field call to add or subtract from a packed decimal field. IMS does the arithmetic for you.

There are two additional field types you can define in your DBD for use with field calls. You can use these field types only on a DEDB or an MSDB:

H halfword
F fullword

IMS also supports addition and subtraction with these field types.

5.1.1.2 How to Use a Field Call

The data language interface function code for a field call is FLD. A simple FLD call will change the value of a field unconditionally. As the field value may change between the call and the commit, you seldom find this kind of FLD call useful.

You can ask IMS to make your update conditional on the value of this field you are updating (or on other fields in the same segment).

You make field call requests to IMS by coding Field Search Arguments (FSAs) which are similar to SSAs, except that you pass them to IMS instead of an I/O area. You also need SSAs to identify the segment to which the FSAs apply. Full details of how to code a field call are in the section“FLD call” in the chapter “Writing DLI Calls for Your Application Program” of *IMS/ESA V5 Application Programming: Database Manager*, SC26-8015.

You then make what is called a **verify** call. If the condition in the verify call is true, we say the call has succeeded. If the condition is false, we say the verify call has failed.

When your program reaches a commit point, IMS reprocesses your verify calls. If any of them fail, IMS backs out your updates and reschedules the transaction with the original message. This happens repeatedly until all the verify calls succeed. (This can be embarrassing if you have accidentally coded a verify call that can never succeed).

IMS intends you to use the verify call to check that a field has not changed value between reading and updating. You can't use it in a situation where it would be normal for the condition to fail. There is an example of such a case in 6.2.3, "Price Movements" on page 71.

locking, duration A field call never returns data to your program. If you must know the exact value of a field, you must read it with one of the DLI get calls. When you are doing this, be careful to use a get-only PCB, otherwise you will lock the field until you reach a commit point.

From IMS Version 5 you can use field calls with any DEDB. We recommend that you use fields calls with MSDBs or VSO DEDBs only. This is because non-VSO DEDB reads usually go to DASD.[4] We assume you are using a field call because you expect several programs to update the data at once. Although your programs do not contend with each other for the data, they each must wait for IMS to read their own copy of the data from DASD. If you have cached DASD controllers, this overhead might be acceptable.

5.1.1.3 An Example Using Field Calls

You may have an application that needs to process several input messages as a single unit, and then perform a completeness check after the last message. Assume that you cannot have a single multisegmented input message, perhaps because of contention problems in the sending application.[5] There is more than one way to deal with such processing.

Process Serially

The simplest way is to process the messages serially and have a flag on the last message as the trigger to perform the completeness check. This approach has the advantage of simplicity and may be necessary if you cannot process the messages in parallel. However, it is not a suitable approach if you need high performance.

Process in Parallel

Another approach is to process the messages in parallel and have a counter to say how many messages there are in total. The application decrements this counter every time it finishes processing a message, and does the completeness check when the counter reaches zero.

The counter, however, is a bottleneck. If you update the counter in the usual way (using DLI GHU and REPL calls), your application still processes serially. This is because each region has to wait for access to the counter in order to update it. IMS locks the counter from the time your application reads it until the commit processing has completed.

---

4 Except if you are reading a segment that happens to be in use already.

5 The sending application might be a BMP that must read many segments from several databases in order to create the messages. This BMP might be forced to issue a checkpoint call (to release resources for other applications) before it can complete a set of messages. In such a case, the BMP could not put all the data into one message because the checkpoint causes IMS to send the message.

Use a Field Call

You need the field call. A field call locks the counter for a short time during commit processing. The short duration of the lock allows your application to run in parallel with only a very small chance of contention.

Using the field call by itself will not completely solve the contention problem in this example. You would have to use an MSDB or a VSO DEDB for the counter because non-VSO DEDB reads usually go to DASD.[4] You would then find your programs waiting for DASD rather than access to the data.

Don′t Expect Too Much
The field call cannot solve all your contention problems. In this example, you still have to read the counter (using a get-only PCB) to find out if it is currently 1 (and therefore the final checking should take place). This is because a failed verify call causes IMS to rerun your transaction. A verify call that always fails will result in your application looping.

5.1.2 Holding Data in Main Storage

The advantage of holding data in main storage is speed. It is faster to read or write data when DASD is not involved.

You get different benefits from holding data in storage with different types of database.

5.1.2.1 Full-Function Databases

If you have enough storage, you can have DLI buffer pools large enough to hold most database records in storage.

Reads benefit from this because there is no need to go to DASD to retrieve data that is already in the buffer pool. If you find that IMS satisfies at least 97% of read requests from the buffer pool, then your buffer pool is large enough. Even if you do not have enough storage to achieve this for all databases, you may be able to achieve it for a few highly active databases by creating a separate buffer pool for them.[6] It may be that your truly active records are always in the buffer pool, but this is difficult to prove.

Writes do not benefit from large buffer pools, because IMS writes updated full-function buffers to DASD at commit time and waits for the write to complete.

In summary, although full-function databases can be fast to read, they are written slowly.

5.1.2.2 Non-VSO DEDBs

A read for a non-VSO DEDB database CI must usually go to DASD, because IMS releases DEDB database buffers at the earliest opportunity.

DEDB writes do not benefit from large buffer pools either, but this is less of a problem because output threads write the updated records to DASD after the commit process has finished.

---

[6] This is likely to be counterproductive if you create the special pool by taking storage away from an existing DLI buffer pool.

In summary non-VSO DEDBs are read slowly, but are fast to write. This is in contrast to full-function databases.

5.1.2.3 VSO DEDBs

VSO DEDBs are the same as non VSO DEDBs except that most reads and writes are to or from a data space rather than DASD. The first read is always from DASD,[7] and IMS writes updated records to DASD at checkpoints or when IMS crosses a threshold.

VSO DEDBs are therefore both read fast written fast.

5.1.2.4 MSDBs

IMS holds MSDB records in storage all the time. Therefore, MSDBs are also read fast and written fast.

5.1.2.5 Buffering Conclusions

MSDBs and VSO DEDBs are faster than other types of database. When speed is crucial, these are the types of database to choose.

DEDBs are more storage efficient than other database types.

Full-function databases are better than non-VSO DEDBs for data you often read but seldom update, provided the data stays in the buffer pool. You can ensure this by reading the data frequently enough or by having a large enough buffer pool. This distinction becomes important only when you cannot use VSO for some reason.

5.1.3 Fast Path Database Buffers and Output Threads

IMS has a single pool of Fast Path database buffers for all Fast Path databases (including MSDBs).[8] All buffers in the buffer pool are the same length, which means that they have to be large enough for the DEDB with the largest control interval. It also means you should be careful about designing DEDBs with a larger than average CI size. We pick up this point later in 6.3.9, “CI Sizes in the Sample Application” on page 78.

You define the buffer pool in the **FPCTRL** stage one macro. You specify the size of each buffer, the total number of buffers, and the number of buffers that IMS should page fix.

You also specify the number of output threads on the FPCTRL macro. An output thread is a separate task that IMS starts. It writes updated DEDB buffers to DASD once IMS has logged the update. This allows the region continue processing. You need to specify enough output threads to prevent the Fast Path database buffer pool from filling with updated CIs.

You may override any of the FPCTRL parameters at IMS start by using the DFSPBxxx member of PROCLIB.

There is a full description of all these macros and parameters in the *IMS/ESA V5 Installation Volume 2: System Definition and Tailoring*, SC26-8024..

---

[7] You can ensure that this first read does not affect your application performance by preloading the database. See 5.2.4.2, “How to Use VSO” on page 55 for details.

[8] Except for programs using HSSP, which each has its own buffer pool.

5.1.3.1 Allocating Buffers to Regions

IMS allocates buffers to a region based on two parameters to DFSRRC00. These parameters are normal buffer allocation (NBA) and overflow buffer allocation (OBA). They set the number of buffers a region may use.

As a program executes, IMS will give it buffers until it tries to exceed the region's NBA. At this point, IMS warns the program (if it is a BMP) by returning an FW status code on the DLI call.

IMS continues to give the program buffers until it tries to exceed the sum of NBA and OBA. At this point, IMS discards all the program's updates and abends it (for message processing programs (MPPs) and interactive fast path programs (IFPs)) or warns it (with an FR status code) for batch message programs (BMPs).

This is a simplification of a much more complicated story. IMS attempts to reuse NBA buffers to avoid using the OBA. If you are interested in the details of this process, refer to the section "Fast Path Buffer Allocation Algorithm" in the chapter "Database Design Considerations for Fast Path" of the *IMS/ESA V5 Administration Guide: Database Manager*, SC26-8012.

5.2 When to Use DEDBs

The art of knowing when to use a DEDB relies on understanding the differences between DEDBs and other databases.

5.2.1 DEDB Areas

You can divide a DEDB into a number of data sets, called *areas*. Each area contains every segment type. The DEDB randomizing routine decides which roots belong in which area. IMS stores all of a root's dependents in the same area.

Most Fast Path commands and utilities operate on an area level, so they do not affect the whole database at once. For example, you can recover one area of a DEDB while the rest of it is in use.

Another reason you might want to use areas is to spread the I/O load across several devices (and hopefully several physical paths in the system I/O configuration).

5.2.1.1 Why You Would Write a DEDB Randomizer

The sample DEDB randomizer (DBFHDC44) spreads roots randomly throughout the database. If you have to stop an area for some reason, the stop will affect a random set of your users. If you write your own randomizer, you can control whom you affect when you need to stop an area.

For example, you might decide to place all data belonging to a particular branch office in one area. If you had to stop that area, only users belonging to that branch would be affected.

5.2.1.2 Tips for Writing a DEDB Randomizer

There are a number of things to bear in mind if you are going to write your own randomizer.

IMS will call your randomizer from several tasks at once. You must ensure your code is reentrant.

Unless you have special requirements, you should write your randomizer to select the area and call DBFHDC44 to select the RAP within the area. When you code this logic, you should use the information IMS supplies. One of the things to consider is the number of root anchor points (RAPs) in each area (areas do not have to be the same size).

If you must write your own RAP selection logic, take care to avoid synonym chaining. You are likely to get contention problems if you have a significant amount of synonym chaining.

When you are writing your area selection logic, the following tips could save you problems in the future:

Enlarging an Area — You can change the size of an area by modifying the AREA macro in your DBD. If your randomizer selects the area, but lets DBFHDC44 choose the RAP, you need only unload and reload the area that changed.***:*** You must use the Database Tools DEDB unload/reload utility or write your own program to perform the unload and reload.

You need to take an image copy of your area as soon as possible after unloading and reloading. This is because the IMS log records for this area cannot be used (they contain RBAs that have now changed).

How to Split an Area — You may find that some areas of your database grow faster than others. Perhaps you chose to split your data by branch office and some branches are unexpectedly much busier than others. You might also want to split a large branch into several smaller ones. In either case you want to split one area into several.***:*** You need to modify your randomizer logic to take account of the new areas in such a way that you affect only the area you are splitting. You then unload and reload that area.

Again you must find a suitable program to do the unload and reload for you, and you must take an image copy afterwards.

Further Help — If you want more information on how to write your own DEDB randomizer, refer to the chapter “DEDB Randomizing Routine” in the *IMS/ESA V5 Customization Guide*, SC26-8020.

5.2.2 Using SDEPs

We met SDEPs in 1.2.1.5, “Sequential Dependent Segments” on page 3. They have these characteristics:

- SDEPs are fast to insert.
- They are slow to retrieve from their parents, but fast to retrieve as a batch.
- SDEPs have some strange properties because IMS stores them separately from their parents.

5.2.2.1 When to Use SDEPs

You would typically use SDEPs when you want to insert data quickly, but do not need to read it again until later. For example you might want to use SDEPs to hold audit records describing sensitive actions the user takes. You would not use SDEPs to hold data for a long time.

5.2.2.2 How to Use SDEPs

Your application inserts SDEPs throughout the online day.

While you can retrieve SDEPs with *get next within parent* (GNP) calls, it is much better to retrieve them in batch using the DEDB SDEP scan utility. This utility reads a range of SDEPs and writes them to a sequential file. You can use this file in later job steps.

After you have finished with the SDEPs, delete them using the DEDB SDEP delete utility.

You can find out more about these utilities in *IMS/ESA V5 Utilities Reference: Database Manager*, SC26-8034.

5.2.2.3 An Example of Using SDEPs

The stock database has an SDEP segment to hold stock transaction details. You can see how we defined the segment in A.2.1, “Stock Database” on page 82.

A.3.2.2, “Deal Closed Program for IMS” on page 103 inserts SDEPs to the stock database whenever we deal in a stock. Hopefully, this is a frequent occurrence, so the insert must be fast. We do not read these segments again until the market closes.

At the close of business, the DEDB SDEP scan utility extracts the SDEPs for A.3.4.2, “Market Close Program” on page 111. The sample JCL for this job also uses the DEDB SDEP delete utility; you can find it in A.3.4.3, “Market Close Job” on page 119.

5.2.3 Using MADS

We introduced MADS in 1.2.1.4, “Multiple Area Data Sets and Record Deactivation” on page 3. Its main characteristics are that IMS

- Keeps many copies of each area.
- Reads from each copy in turn.
- Tolerates read failures unless all copies of the CI are bad.
- Writes to every copy at once.
- Tolerates write failures.
- Allows you to repair the database while still using it.

5.2.3.1 When to Use MADS

You use MADS to ensure that I/O errors do not affect a database. Normally two copies of each area would be sufficient, but you can have up to seven copies if you wish (provided you do not exceed the limit of 240 area data sets per database).

Obviously, using MADS is costly because you have several copies of the data. There is also a cost at execution time because IMS has to update several copies of the database simultaneously. The transactions using the DEDB do not notice

the extra I/O, because the output threads handle it asynchronously. You should use MADS only when you can justify the extra DASD cost.

If you have very valuable data you should be able to justify MADS terms of the time that it saves if you ever need to repair the database. Without MADS, you could not use an area while you were recovering it. With MADS you can use the DEDB area data set create utility to repair an area while it is still in use.

5.2.3.2 How to Use MADS

To use MADS, tell IMS that you are using it via DBRC. If you have more than one ADS entry per DBDS entry, then IMS knows you are using MADS. There is an example of defining a MADS database to DBRC in A.2.6, “DBRC Definitions” on page 86.

When you add a new area data set to an existing database, you use the DEDB area data set create utility. The utility runs in parallel with online activity. You would also use this utility to create an error-free area data set if any or all of your existing area data sets have errors.

If you suspect differences between your area data sets, you can use the DEDB area data set compare utility to find out.

5.2.4 Using VSO

We introduced VSO in 1.2.1.9, “Virtual Storage Option (VSO)” on page 4. VSO has the following characteristics:

- Once IMS has read a CI from a VSO database, IMS keeps the CI in a data space indefinitely.
- IMS does not store VSO SDEPs in the data space.
- VSO databases lock at segment level rather than CI level.

5.2.4.1 When to Use VSO

Use VSO for your most frequently used databases or those for which fast access is crucial. It is also good for data you update frequently, even if several applications want to update the same field at the same time.

It is worth pre-opening the majority of your VSO databases, to take the overhead of opening the database away from the first application to execute. If a database is active enough to justify being VSO, is likely to be active enough to justify early opening.

It is worth preloading a VSO database if it is very active unless it is very big. This is because the bulk of the database will end up in the data space anyway, so you might as well save your applications having to do it.

If you have a database whose reference pattern varies significantly over time, you may find it useful to free the VSO space that database is using and start reading again from DASD. You can do this with the /VUNLOAD (virtual unload) command.

For example, a financial application might need quick access to data entered today, but not to data entered yesterday. If you issued a /VUNLOAD AREA

followed by a /START AREA[9] overnight, you would prevent the VSO data space from filling with unimportant data.

5.2.4.2 How to Use VSO

You tell IMS that you want to use VSO by adding parameters to the DBRC DBDS entry. This means that you don't have to use VSO for all the areas in your DEDB if you don't want to. There are three parameters to control the three VSO options:

- VSO defines the database as VSO (NOVSO does the opposite).
- PREOPEN says that IMS should open the database at IMS start or when you start the area. NOPREO tells IMS to wait until the area is first accessed.
- PRELOAD tells IMS to load the entire database into a data space at IMS start or when the you start the area. NOPREL tells IMS to load the CIs into the data space as they are accessed.

There are sample VSO database definitions in A.2.6, "DBRC Definitions" on page 86 and full details of DBRC command syntax in the *IMS/ESA V5 Utilities Reference: Database Manager*, SC26-8034.

5.2.5 Using HSSP

We talked about HSSP in 1.2.1.10, "High Speed Sequential Processing" on page 5. It is a fast processing option that can only be used

- By non-message-driven BMPs
- On DEDBs in strict hierarchical sequence
- By itself (not in conjunction with other HSSP jobs or Fast Path utilities).

Use HSSP for only those programs that conform to its restrictions, because you get better performance. You also get better performance in IMS Version 5 because of enhancements it has made to HSSP.

Consider using the option to let HSSP take an image copy while it is running. This will save you time if you would normally take an image copy after your program finishes. (HSSP is clever enough not to log updates for a database it is copying).

There are a number of things you must know to get your program to use HSSP.

5.2.5.1 PSB and PROCOPT

You declare your intention to use HSSP by specifying PROCOPT=H in your PSB. You can have only one such PCB per database in your PSB. Processing option H implies options G (get) and P (position). When you are using HSSP, you must take a commit point when you get a GC status code.

We explain PROCOPT=P and the GC status in a little more detail in 5.2.7.3, "Path Calls, Processing Option P, and GC Status" on page 58.

---

[9] If you did not issue the /START AREA, IMS would not store any data in the data space. The /VUNLOAD disables VSO options until the next /START AREA or IMS restart.

5.2.5.2 How to Take an Image Copy with HSSP

To request an image copy, you use the SETO keyword in the DFSCTL file. There is an example of this in A.3.4.3, “Market Close Job” on page 119.

If you want HSSP to take an image copy for you, your DEDB and the image copy data sets must be defined to DBRC. As a result, HSSP writes lots of messages to the master and secondary master terminals when you are using the image copy option.

You can also use the SETO keyword to

- Disable HSSP.
- Restrict the image copies to certain areas of your DEDB.
- Specify what action IMS should take if the image copy fails.

There is more information about this keyword in the section “Specifying HSSP Control Statements” in the chapter “Tailoring the IMS System to Your Environment” of *IMS/ESA V5 Installation Volume 2: System Definition and Tailoring*, SC26-8024.

5.2.5.3 Fancy Processing with HSSP (Specifying a Range)

You can use the SETR keyword to specify a list of DEDB areas that HSSP should process. This list need not contain all the areas, nor need they be in the same order as in the database definition (DBD). HSSP processes the areas in the order that you list them.

You specify the SETR keyword in the DFSCTL file. There is more information about this keyword in the section “Specifying HSSP Control Statements” in *IMS/ESA V5 Installation Volume 2: System Definition and Tailoring*, SC26-8024.

5.2.5.4 Other Things You Should Know about HSSP

HSSP uses its own buffer pool. In fact, HSSP allocates buffers equal to the size of three units of work. This means it can have more buffers than a normal BMP without affecting your Fast Path database buffer pool.

HSSP locks at unit of work level. This can be a problem if it runs when your database is busy.

5.2.6 Using Subset Pointers

Subset pointers are a special kind of pointer that your application can set and use. You can have up to eight subset pointers per segment. IMS holds the pointers in the parent segment, so you cannot have subset pointers on a root or an SDEP segment.

5.2.6.1 When to Use Subset Pointers

You can use subset pointers for whatever purpose you like, but you will probably find them most useful if you have long twin chains. Using a subset pointer you can keep your place in a twin chain, indefinitely if necessary. When you want to reestablish position in the chain, IMS has only to read the parent and then the segment you want.

Using subset pointers can save a lot of processing and I/O, because the only other way to reestablish position within a twin chain is to read all the preceding

twins.[10] Even if you were to use the segment key to reestablish position with one call, IMS would still have to read through the twin chain from the beginning until it found the segment you wanted.

5.2.6.2 How to Use Subset Pointers

Before you can use a subset pointer, it must be defined in the DBD and you have to access it through your PSB (by using the SSPTR parameter in the SENSEG macro).

You use the following SSA command codes to manipulate subset pointers.

Rn Read subset pointer n to establish position.
Zn Set subset pointer n to zero.
Mn Set subset pointer n to the next segment.
Sn Set subset pointer n to the current segment.
Wn Set subset pointer n to the current segment, but only if subset pointer n is currently zero.

You can use most of these codes in combination. Please refer to the section "Command Codes" in the chapter "How Your Application Program Works with the IMS Database Manager" of *IMS/ESA V5 Application Programming: Database Manager*, SC26-8015, for further details and examples of use.

5.2.7 Similar to Full-Function but Different

There are a number of minor differences between DEDBs and full-function databases which can confuse you if you are not expecting them.

5.2.7.1 Reorganization

The high-speed DEDB direct reorganization utility reclaims fragmented DASD space only. ***You cannot use the high-speed DEDB direct reorganization utility to change a DEDB's structure***.

If you want to change the structure of a DEDB you must use the Database Tools DEDB unload/reload utility or write your own program. This makes it more difficult to add segment compression to a DEDB. If you want segment compression, add it at the design stage.

The high-speed DEDB direct reorganization utility runs in parallel with online work and locks at unit of work level. It reorganizes only direct dependent space (and not SDEP space), hence the word "direct" in the name.

From IMS Version 5, the high-speed DEDB direct reorganization utility uses HSSP, hence the words "high speed" in the name. If you specify enough buffers, this utility will read ahead asynchronously. Refer to *IMS/ESA V5 Utilities Reference: Database Manager*, SC26-8034 for details.

---

[10] You can easily reestablish position at the end of a twin chain if you use Physical Child Last pointers and your segments are unkeyed. See 6.3.10, "Long Twin Chains" on page 78 for an example of this.

5.2.7.2 Multiple Positioning

IMS automatically uses multiple positioning in a DEDB without making you declare it in your PSB. This is in contrast to a full-function database where you must ask for multiple positioning if you want it.

Multiple positioning allows you to keep position at several points in a database hierarchy simultaneously. It affects whether or not IMS will find a segment when you use a GNP call. There is a detailed description of this topic in "Using Multiple Positioning" in the chapter "Using Multiple Processing" in *IMS/ESA V5 Application Programming: Database Manager*, SC26-8015.

5.2.7.3 Path Calls, Processing Option P, and GC Status

IMS automatically allows path calls for DEDBs, without making you declare it in your PSB.

In fact, DEDBs use PSB PROCOPT=P for something completely different. If you specify this option for a BMP, IMS will return a GC status code whenever your program crosses a unit of work boundary. This is a convenient time to take a checkpoint, because a unit of work comprises a (small) number of database records.

If you are using HSSP, you must take a commit point when you receive a GC status.

5.2.7.4 Segment Pointers

You can specify what pointers IMS should provide for each segment in a full-function database. You do this with the POINTER= parameter of the SEGM macro. You cannot do this with a DEDB, because this parameter is not allowed.

This restriction means you cannot have

- Twin backward pointers. These are useful if you have a lot of delete activity on a long chain, because they help IMS maintain the twin forward chain.
- No twin pointers at all (useful if you know there cannot be more than one occurrence of a segment).
- Hierarchical pointers. This is not a great loss because such pointers are useful only in special circumstances.

You can have physical child last pointers (because you do not specify these with the POINTER= parameter). These are useful for long twin chains, provided IMS can be sure that you always insert new segments at the end of the chain. Our sample application uses this type of pointer; see 6.3.10, "Long Twin Chains" on page 78 for an explanation.

5.2.7.5 Fixed-Length Segments

A fixed-length segment in a full-function database has no length field. A fixed-length segment in a DEDB, however, does have a length field, in the segment prefix. Your application cannot see this field and you know it is there only if you print the CI (or look at an image copy).

Note: Variable-length segments are the same for DEDBs and full-function databases. In this case, the length field is part of the segment data.

5.2.8 When to Use Full-Function Databases

There are times when the restrictions on DEDBs become too onerous and you should consider using full-function databases.

5.2.8.1 Logical Relationships and Secondary Indexes

Full-function databases can make it easier to implement logical relationships and secondary indexes, as there is no support for either of these functions in Fast Path. However, most logical relationships are simple and therefore easy to emulate with application logic. For example, the sample application has relationships between its databases. See 6.3.8, “Emulating Logical Relationships” on page 77 for details.

More complicated logical relationships are harder to emulate, especially those that combine several databases into one logical database or invert the database structure. In this case it is easier to use IMS logical relationships.

Inquiry applications often make use of a secondary index to retrieve data quickly without using key sequence. There is a trade-off between the cost of maintaining the index (when IMS updates the database) and the saving when IMS reads the database using the index. You can code your application to maintain its own index, but it is tedious and error prone.

In summary, if you need a simple logical relationship, you should be able to emulate it satisfactorily with DEDBs. If you need a complicated logical relationship or a secondary index, use a full-function database.

5.2.8.2 Roots in Key Sequence

If your application needs to process root segments in key sequence, you will be able to use a DEDB only if you can code your DEDB randomizer to assign root anchor points in that sequence. This is unlikely to be practical unless you have a good idea of how many root segments there are and what keys they have.

If you cannot code your randomizer in such a way, a HIDAM database is likely to better suits your needs.

5.2.8.3 Fast Path Database Buffer Pool Saturation

You are limited to 9999 Fast Path database buffers. IMS uses the same buffer pool for all DEDBs and MSDBs. If you have a lot of Fast Path databases, you may find this limit restricting.

In either case, you should consider converting some databases to full-function.[11] You should convert your reference databases (those you read frequently but seldom update). We explain the reason for this in 5.1.2.5, “Buffering Conclusions” on page 50.

[11] Converting to VSO will not help because IMS uses the Fast Path database buffer pool to hold VSO CIs while they are in use.

5.2.8.4 Short Database Records

Some databases have very short records, often only root segments. If you have such a database, you will probably run into problems with DEDBs. If you have a database with short records, IMS will seldom use the overflow parts. You are effectively wasting the DASD space.

The main problem arises because non-VSO DEDBs lock at CI level. If you use a non-VSO DEDB in this case, you face a choice between wasting space and locking. You waste DASD (and buffer pool) space if you store only one root per CI. However, if you store many roots in a CI, you lock locking many root segments together with the one you are accessing, resulting in contention problems.

You do not avoid the problem by using a VSO DEDB, although VSO DEDBs lock at segment level, because “segment level locking” is a misnomer. That is, VSO and full-function databases lock a root segment and all of its children. In the case of VSO and HDAM, this includes the RAP ***and all the roots chained from it***.

The only solution is to have many anchor points per CI and segment level locking. This is possible only with a full-function database.

There is a good example of this dilemma in our sample application. See 6.3.1, “Currency Database” on page 73 for details.

5.2.8.5 Miscellaneous

If you need more than 127 segment types, you must use a full-function database.

DEDBs must be VSAM. If for some reason you cannot use VSAM, you would have to use an OSAM full-function database.

5.2.9 When to Mix Full-Function and DEDBs

You should not worry about mixing DEDBs and full-function databases.. There are a few things to bear in mind when mixing, but no real pitfalls.

Our sample application contains both full-function databases and DEDBs.

5.2.9.1 The Advantage of Mixing

Not every database in your application is likely to have the same characteristics. As we have seen, each type of database has its own strengths and weaknesses. The skill of the database designer is to know what type of database to use in each particular case. If your design work leads you to mix DEDBs and full-function databases, then it is undoubtedly the right thing for you, if the disadvantage is acceptable.

5.2.9.2 The Disadvantage of Mixing

There is only one disadvantage. As we noted in 1.2.3, “Mixing Fast Path and Full-Function Databases” on page 7, IMS writes full-function database updates to DASD at commit time and waits for the write to complete. This is in contrast to a DEDB where an output thread does the updates later.

Mixing both types can slow your application, but you would have to have a high transaction rate before the slowing would become noticeable.

5.3 Designing a DEDB

Once you have decided to create a DEDB and decided which options you are going to use, you should get some idea of the likely segment distribution. This is important because it helps you calculate the average database record length. This in turn helps you decide two things that you cannot easily change: the CI size and the unit of work size.

5.3.1 Calculating the Average Database Record Length

When you calculate the average database record length, you have to include the IMS overheads. There are overheads at segment and CI level.

All IMS database segments have a prefix that contains information IMS needs to know, but your application doesn′t. In a DEDB, the minimum prefix is 6 bytes:

- For the segment code, 1 byte
- For the segment type, 1 byte
- For the twin forward pointer, 4 bytes.[12]

Other factors add to the prefix:

- For an SDEP pointer, 8 bytes
- For every other kind of pointer, 4 bytes per pointer
- For a length field, if the segment is fixed length, 2 bytes.[13]

There is also an overhead for each DEDB CI. This is 15 bytes in an SDEP CI and 21 bytes in other CIs. If you want to know the layout of a CI in detail, please refer to “Parts of a DEDB Area” in the *IMS/ESA V5 Administration Guide: Database Manager*, SC26-8012.

We show how we calculated the average database record length for each of our sample application databases in 6.3, “Databases” on page 73.

5.3.2 Picking a CI Size

You should bear the following factors in mind when deciding a CI size for your DEDB:

- All Fast Path databases (including MSDBs) use the same buffer pool. All buffers in the buffer pool are the same length, which means that they have to be large enough for DEDB with the biggest CI size. If most DEDBs have a CI size smaller than the maximum, you waste buffer space.

  There is an example of this in our sample application; see 6.3.9, “CI Sizes in the Sample Application” on page 78.

- A DEDB has only one root anchor point per CI, so you should not have more than one root per CI.[14] There is therefore no point to having a large CI size if the average database record is short.

- DEDB locking is at CI level, except for VSO, which locks at segment record level.

---

[12] You cannot suppress this pointer; see 5.2.7.4, “Segment Pointers” on page 58.

[13] This is different from a full-function database; see 5.2.7.5, “Fixed-Length Segments” on page 58.

[14] Even in a well randomized database, there will be a small amount of synonym chaining, so some CIs will have more than one root. The point is that most CIs will have only one root.

This is another reason why you should not plan to have more than one root per CI.

- Media manager restricts the CI size to 0.5 KB, 1 KB, 2 KB, 4 KB, 8 KB, 12 KB, 16 KB, 20 KB, 24 KB or 28 KB.

The following items discuss ways to control the database record length, so that IMS can use DASD space (and hence buffer pool space) more effectively. These tricks of the database designer′s trade are outside the scope of this book. However, you will find more information the *IMS/ESA V5 Administration Guide: Database Manager*, SC26-8012 and the *IMS/ESA V5 Utilities Reference: Database Manager*, SC26-8034.

In controlling database record length,

- You cannot use multiple data set groups as a way of controlling the database record length. DEDBs do not support multiple data set groups.
- You can use segment compression routines to control the database record length. Now is the time to decide on that, because it is tedious to add segment compression as an afterthought.

Having chosen the CI size, you need to inform IMS of the choice, using the SIZE parameter of the AREA macro.

5.3.3 Picking a Unit of Work Size

Once you have picked a CI size, you must decide how many CIs make a unit of work. A unit of work has two parts, each comprising a number of CIs.

The base part contains CIs with root anchor points. IMS uses these CIs to store root segments and as many of their dependents as will fit. IMS stores SDEP segments in a different part of the data set.

The dependent overflow part contains CIs that IMS uses when a base CI runs out of space. IMS assigns one of the dependent-overflow CIs to be a logical extension of a base CI. If another base CI runs out of space, IMS assigns it a different dependent-overflow CI (assuming there are any left). You can find out what happens when the dependent-overflow CIs are all used up by referring to 5.3.4, “Designing an Area” on page 63.

Another way to think of a unit of work is as a number of roots plus all their direct dependents. Understanding that a root′s dependents belong to the same unit of work helps you understand why IMS encourages you to checkpoint at the end of a unit of work.

You should bear the following factors in mind when choosing a unit of work size.

- The high-speed DEDB direct reorganization utility and HSSP programs lock an entire unit of work at a time. You should therefore choose a smaller rather than a larger size in order to reduce contention with other applications.
- Batch programs can ask for notification when they reach the end of a unit of work, because this is a good time to take a commit point. (See 5.2.7.3, “Path Calls, Processing Option P, and GC Status” on page 58 for more details.) If you to choose too small a unit of work size, your BMPs will be forever checkpointing.

- Choosing a very small unit of work makes it difficult for IMS to manage the dependent overflow part efficiently. The minimum dependent-overflow size is one CI, so if your unit of work is small the overflow will be a large proportion of the total.

If you still cannot decide on a unit of work size, you won't go far wrong if you pick a number around ten.

Once you have decided the size of a unit of work, code it in the UOW parameter of the AREA macro.

5.3.4 Designing an Area

Now you have chosen a UOW size, you are ready to decide the size of the area. You allocate area space in units of work, and there are three parts of an area to consider.

The root addressable part contains UOWs with the layout we describe in 5.3.3, "Picking a Unit of Work Size" on page 62.

The independent overflow part contains a pool of CIs for use when a UOW has exhausted its dependent-overflow CIs. IMS assigns the independent overflow CIs to a base CI on an individual basis. If you fill up your entire independent overflow part, there is a way to expand it. Please see 2.1.9, "Expansion of DEDB Areas" on page 12 for details.

The sequential-dependent part comprises the rest of the data set (up to the amount of space you allocate). IMS uses this to store SDEP segments.

You should allow enough space for reasonable data growth. Use the ROOT parameter of the AREA macro to define the size of the root-addressable and independent-overflow parts of the area.

5.3.5 Defining Your DEDB to DBRC

It is a good idea to define your DEDB to DBRC. You must define your DEDB to DBRC if you use any of the following:

- MADS
- VSO
- The image copy option of HSSP

If you define a DEDB to DBRC, you do not need to code DFSMDA macros for it. DBRC contains all the information IMS needs to allocate the database.

There are some sample definitions in A.2.6, "DBRC Definitions" on page 86. You can find the full syntax of DBRC commands in the *IMS/ESA V5 Utilities Reference: Database Manager*, SC26-8034.

5.3.6 Initializing a DEDB

You must initialize a DEDB before you can use it. These are the steps you follow.

1. Delete and define the data sets. The DBDGEN output contains suggested IDCAMS parameters.
2. Run the DEDB initialization utility

3. Take an image copy.
4. Start all the database areas.

Your DEDB is now ready for you to add the data. You do not need a load mode PSB to add data to a DEDB.

5.4 When to Use MSDBs

You should bear two things in mind when reading this section. First, if you have IMS Version 5, all MSDB features are available to VSO DEDBs. Second, IMS will not enhance its support for MSDBs and you should seriously consider migrating your MSDBs to VSO DEDBs.

MSDBs have two main advantages over other types of database:

- They are good for storing data that you constantly update, because of the special features of the field call. There is a discussion of field calls in 5.1.1, “Field Calls” on page 47.
- They are very fast to process because they are held in storage.

Typically, you would chose to use an MSDB when one of these advantages is crucial to the success of your application, and you could not use a VSO DEDB for some reason.

5.5 Designing an MSDB

There are four types of MSDB, but only one is really useful. This is the non-terminal-related database without terminal keys. The other types of MSDB are less useful because they have one segment for each Fast-Path-capable terminal in your IMSGEN. If you are using ETO, you won't have any terminals in your IMSGEN.

You can read about the types of MSDB in the section “Main Storage databases” in the chapter “Designing a Fast Path Database” in the *IMS/ESA V5 Administration Guide: Database Manager*, SC26-8012.

5.5.1.1 MSDB Design Restrictions

An MSDB can only contain fixed-length root segments.

You cannot insert segments into or delete segments from a non-terminal-related MSDB. You can update such a database with either replace or field calls. If you want to change the number of segments in such a database, you must use the MSDB maintenance utility.

5.5.1.2 MSDB View of Updated Data

You do not see any changes you make to an MSDB until after your program has committed. This is true whether you use insert, delete, replace, or field calls. The MSDB is different from any other kind of database, where a program can see its own changes before commit.

For example, if your program replaces a segment and then rereads that segment, your program sees the old field values.

5.6 Converting an MSDB to a DEDB

IMS Version 5 helps you convert your MSDBs to DEDBs by extending all the features of MSDBs to DEDBs. There is also an MSDB-to-DEDB conversion utility.

5.6.1.1 No Code Changes

You can convert your application from using MSDBs to using DEDBs without having to change the source code. However, you must change your PSB if your application relies on the MSDB view of updated data. You can request the MSDB view by specifying VIEW=MSDB on any DEDB PCB.

5.6.1.2 Advantages of Converting

DEDBs have many more features than MSDBs (for example, you can delete segments from a DEDB). Furthermore, it is much easier to recover a DEDB than an MSDB because DBRC does not support MSDBs.

Finally, no enhancements to MSDBs are planned. IBM may drop support for MSDBs in the future.

5.6.1.3 Conversion Process

There are a number of steps in the conversion process:

1. Write a DEDB version of the MSDB DBD.
2. Create an MSDBINIT data set (using the MSDB dump utility).
3. Convert the MSDBINIT data set into a DEDB reload file (using the MSDB-to-DEDB conversion utility).
4. Delete, define and initialize your DEDB in the usual way. (See 5.3.6, “Initializing a DEDB” on page 63 if you are unfamiliar with this process.)
5. Load your DEDB from this file using the Database Tools DEDB unload/reload utility.

This is the reverse process, in case you need it.

1. Unload your DEDB using the Database Tools DEDB unload/reload utility.
2. Sort the unload file into key sequence.
3. Convert the DEDB unload file into an MSDBINIT data set (using the MSDB-to-DEDB conversion utility).
4. Merge this MSDBINIT with your existing MSDBINIT data set (using the MSDB maintenance utility).

You can find full details of these processes in *IMS/ESA V5 Utilities Reference: Database Manager*, SC26-8034.

5.7 Using the Expedited Message Handler

We introduced EMH in 1.3, “Expedited Message Handling (EMH)” on page 7. Its main features are as follows:

- Messages must be single segment, response mode and to or from a terminal.
- Scheduling is first-in, first-out (FIFO).
- EMH uses special IMS Fast Path (IFP) regions.
- You cannot use IMS command calls with EMH.

5.7.1 When to Use EMH

You should use EMH only when its restrictions are not important to you *and* you need the small performance benefit EMH can give. This is most likely to be the case when all of your application uses only MSDBs or VSO DEDBs (or has no databases at all).

Most other applications won't notice the difference between EMH and normal message processing because the number of database I/Os will usually be much larger than the cost of message handling.

5.7.2 How to Use EMH

There are three areas in your system that will be effected when you start to use the expedited message handler:

- Stage one macros
- IFP regions
- Fast Path input edit/routing exit (DBFHAGU0)

5.7.2.1 Stage One Macros

Applications fall into one of three categories

Fast Path exclusive	Must use EMH You define an application as being Fast Path exclusive with the FPATH=YES parameter on the APPLCTN macro.[15]
Fast Path potential	Can use EMH You define an application as being Fast Path potential with the FPATH=YES parameter of the TRANSACT macro, provided you have not specified FPATH=YES on the APPLCTN macro.
Non-Fast-Path	Cannot use EMH This is the case when you have not specified FPATH=YES on either macro.

You must also assign a routing code to a Fast Path Application, either explicitly using the RTCODE macro or implicitly using a TRANSACT macro.

If you want to run a Fast Path potential application in an IFP region, you must define at least one Fast Path exclusive application. IMS will permit an IFP region to start only if you have defined its PSB as FPATH=YES.

It is a good idea to generate a Fast Path potential copy of a Fast Path exclusive application. This allows it to run (without EMH) if there are no IFP regions available. If you don't do this, IMS will reject input messages for your application.

There is an example of generating a Fast Path potential transaction in A.4, "Stage One Macros" on page 125. Full details of the stage one macros are in the chapter "Macros" of the *IMS/ESA V5 Installation Volume 2: System Definition*

[15] If you specify FPATH=size on the APPLCTN macro, that implies FPATH=YES.

and Tailoring, SC26-8024. There is a discussion in “Defining Fast Path transactions” in the chapter “Designing Your System” of the *IMS/ESA V5 Administration Guide: System*, SC26-8013.

5.7.2.2 IFP Regions

IFP regions are more similar to BMP regions than to regions. An IFP region executes a single wait for input program, which has at least one routing code associated with it. IMS uses the routing code to decide which region should process a message.

The IFP region procedure is IMSFP in PROCLIB.

5.7.2.3 Fast Path Input Edit/Routing Exit (DBFHAGU0)

IMS calls this exit for every Fast Path potential or Fast Path exclusive message. The exit allows you to:

- Edit the message
- Change the routing code (and so change the region that will process the message)
- Decide whether Fast Path potential messages will use EMH or not. The default exit sends all Fast Path potential messages to the normal message queue process.

If you have Fast Path potential transactions, you should code the exit to end with return code 16. This causes IMS to schedule a Fast Path potential transaction in an IFP region if one is available or, if not, in an MPP region.

For further details, refer to the chapter “Fast Path Input Edit/Routing Exit (DBFHAGU0)” in the *IMS/ESA V5 Customization Guide*, SC26-8020.

Chapter 6. Sample Application

We chose a stock market application because Fast Path has a number of features suitable for such an application:

- Sequential dependent segments can hold audit records for later batch processing.
- All the databases to be updated frequently are VSO.
- Field calls can be used in several places to reduce contention.
- Online reorganization in Fast Path can reduce outages.
- Multiple area data sets guard important data against loss.
- HSSP can be used in several places.

The international nature of the stock market lends itself to a distributed application. Again Fast Path is useful because of some of it's characteristics.

First, stocks are traded around the globe day and night. Continuous availability is a must and Fast Path has features to help with this.

Widely varying demand is another characteristic of distributed applications. Such an application has to tolerate both high and low demand; Fast Path can do this.

6.1 Outline

The sample application helps a broker trade in stocks throughout the world. We assume that the broker has offices in all the major stock exchanges. To make the application simpler, however, we have chosen only three exchanges: London, New York, and Tokyo.

6.1.1.1 Presumed Central Stock Exchange System

We assume that each market has a central stock exchange system which, among other things, provides stock prices and tracks stock deals.

We assume that the individual stock exchange authorities have implemented their central systems in different and incompatible environments. This means that the central systems do not communicate with each other, so that our application must handle international communications.

In reality, the communication method between the sample application and the central stock exchange system would be different in each case. For simplicity, we assume an asynchronous LU6.2 interface. Although the interface is asynchronous, it must be fast, because stock prices are only current for a short time.

We simulate the barest minimum of the central stock exchange system to provide a framework for our sample application to run in.

Figure 7 on page 70 shows the flow of data.

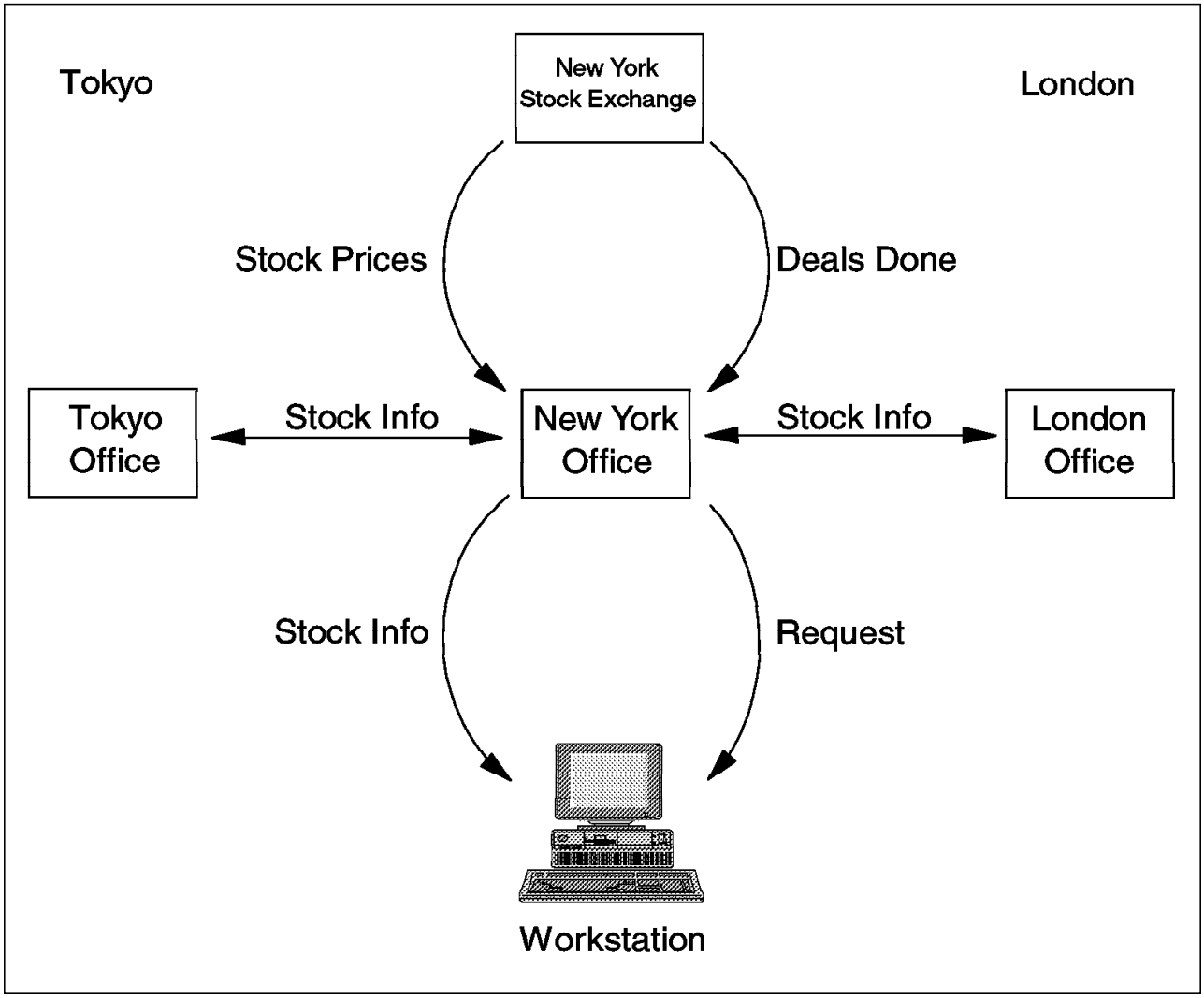


Figure 7. Stock Market Application

6.2 Functions

The sample application has IMS, TSO, and workstation components. The mainframe components are installed in each broker's office. The workstation components are installed in each of the offices and on portable computers for use outside the office.

The sample application has a number of functions. We chose to write the functions in REXX using common programming interface for communications (CPIC) calls because this made them

- Quicker to develop
- Easier for the reader to follow
- Portable between environments.

6.2.1 Stock Inquiry

Stock inquiry is a distributed function: a workstation or TSO user converses with an IMS transaction. The user asks for information and waits for the IMS transaction to retrieve it.

There are a number of subsidiary functions:

- Price quote — preparatory to trading
- Stock holdings — preparatory to selling
- Trends — The workstation can display this information graphically.

If the stock trades on a remote market, the stock inquiry IMS transaction initiates an APPC conversation with a remote copy of itself to retrieve the data. It automatically converts stock prices into the local currency. Also, since a stock may trade on more than one market, it is possible that the stock inquiry will return more than one set of stock information.

This program emulates a logical relationship from the stock and index databases to the market and control databases. See 6.3.8.1, “Links for Market Data” on page 78 for more information on the nature of this link.

The sample code for this application is in A.3.1, “Stock Inquiry Functions” on page 88.

6.2.2 Deal Closed

This is a distributed function: a workstation or TSO user informs an IMS transaction that a deal has been done, presumably by telephone with another broker. The IMS transaction logs the deal, updates the quantity of stock held, and informs the central stock exchange system so that it can amend the stock price if necessary.

We expect a high volume of deals, so we chose an SDEP segment for the logging segment because they are quick to insert. We use a field call to update the stock quantity because we don′t want unnecessary contention. This program uses a CHNG call to specify APPC parameters. The code for this is in A.3.2.3, “APPC Change Call” on page 105, because more than one program uses it.

The sample code for this application is in A.3.2, “Deal Closed Functions” on page 103.

6.2.3 Price Movements

This is a background IMS function, triggered by the local stock exchange system.

The application updates the current stock price and amends the history of today′s movements. If this stock is a member of a stock index, IMS modifies the index to reflect the new price and the number of shares traded.

This program updates the fields that hold the highest and lowest values.

This program also updates a stock index by emulating a logical relationship between the stock and index databases. See 6.3.8.2, “Link between Stocks and Indexes” on page 78 for more details about this link.

The sample code for this application is in A.3.3, “Price Movement Functions” on page 108.

6.2.4 Market Close

This is a background IMS function. We implemented it as a BMP, triggered by the local stock exchange system when it closes. The Market Close program has the following functions:

- It notifies other branches that this market has closed, so that they do not try to trade stocks on this market. It does this by using an APPC change call.
- It interfaces transaction details to the central system for account settlement. A previous job step extracts these details (they are in an SDEP segment) and a subsequent job step deletes them.

  The Market Close program remembers every stock it encounters in this file. This could potentially mean a lot of data, but it has the advantage that we can then process the stock database sequentially and make the program eligible for HSSP.
- The Market Close program reconciles the quantity of stock held with the day's transactions.
- It summarizes today's movements for each stock and updates current history with the summary.
- It recalculates each stock index by totaling the closing prices of every stock in the index.

  The Market Close program remembers every Index it encountered on the stock database. That way, we can process the index database sequentially and therefore qualify for HSSP. There are only a few Indexes, so this not likely to cause a problem.
- The historical data to archived. The sample code deletes the old data. A more sophisticated function is planned that would move the data to another database.

This is a sequential process that benefits from HSSP.

The sample code for this application is in A.3.4.2, "Market Close Program" on page 111 and the sample job is in A.3.4.3, "Market Close Job" on page 119.

6.2.5 Market Open

This background mainframe function is triggered by the local stock exchange system. It notifies other branches that this market has opened. This program uses a CHNG call to specify APPC parameters. The code for this is in A.3.2.3, "APPC Change Call" on page 105, because more than one program uses it.

The sample code for this application is in A.3.5, "Market Open Functions" on page 119.

6.2.6 Data Entry and Correction

A complicated application like this would have extensive data entry and correction functions. We relied on DFSDDLT0 to provide these functions.

6.3 Databases

There are seven databases in this application. Most are VSO because we update them frequently. We pre-open all the VSO databases so that the first user of the day will not suffer this overhead.

Some of the VSO databases are small. We preload them because we want all their records to be in storage all the time.

6.3.1 Currency Database

Currency conversion is fundamental to international stock trading. We choose to simplify this problem by using fixed exchange rates, read from the currency database.

6.3.1.1 Contents

The currency database contains only fixed-length root segments. There is one root per currency. You can find the DBD in A.2.4, “Currency Database” on page 85.

It has an average database record length of 18, calculated as follows:

- 6 bytes for the fixed part of the segment prefix
- 2 bytes for a length field (the segment has a fixed length)
- 10 bytes of data.

We therefore chose a small CI size (1 KB, although 0.5 KB would do just as well).

6.3.1.2 Design Considerations

This database is a good example because it has conflicting requirements. Although we have not coded it, you can imagine that a currency application continuously updates this database with the latest information. Our applications would have to read the database in the midst of all this update activity. This makes the currency database a good candidate for VSO.

On the other hand, the currency database records are very short. If we choose VSO, this limits us to one root per CI[16] and wastes a lot of DASD (and buffer pool) space.

If we choose HDAM for the currency database, we have the problem that IMS will lock the segments while the Currency Application updates them. This will degrade our performance.

In the end, we chose VSO because wasted DASD was the lesser of the two evils, given that this is a small database.

The currency database is small and highly active, so we decided to preload it.

[16] To avoid contention problems. See 5.2.8.4, “Short Database Records” on page 60 for an explanation of why this is so.

6.3.2 Control Database

This database holds miscellaneous reference information.

6.3.2.1 Contents

The control database contains three root segments

- The name of the local market (so we can tell whether a stock or index is local or not).
- Details of the local stock exchange system. These include LU6.2 parameters.
- The number of days′ that historical data should remain on the stock and index databases.

You can find the DBD in A.2.5, “Control Database” on page 85.

This database contains 76 bytes of data, calculated as in Table 1.

Table 1. Calculating the Total Size of the Control Database

Segment Name	Data Length	Prefix Length	Total Length
Local market	8	6	14
Stock exchange	12	6	18
Retention criteria	38	6	44
Total			76

6.3.2.2 Design Considerations

This database is HDAM because it fulfills two of the criteria mentioned in 5.2.8, “When to Use Full-Function Databases” on page 59.

It has database records so short that the entire database would easily fit into one CI.

Our programs read the control database frequently but seldom update it, so seldom that we have not yet coded the update program.

Our programs are careful to read this database once only (and not once per message) since we do not expect the contents to change.

6.3.3 Market Database

This database contains information about copies of this application in other offices around the world.

6.3.3.1 Contents

This is a root-only database with one root per office. It contains LU6.2 parameters and transaction codes. The average database record length is 66, comprising

- 6 bytes for the fixed part of the segment prefix
- 60 bytes of data.

Again, we chose a small CI size (1 KB, although 0.5 KB would do just as well).

6.3.3.2 Design Considerations

Like the control database, the market database is HDAM because it fulfills two of the criteria mentioned in 5.2.8, “When to Use Full-Function Databases” on page 59.

It also has short database records, short enough that the entire database would fit into one CI.

Our programs read the market database frequently but never update it.

6.3.4 Index Database

This database holds details about a stock market index.

6.3.4.1 Contents

If the index is local, the database contains the current and historical values of that index. If the index is remote, the database contains the name of the market where the values are held. In a full-function database, this would be a logical relationship, see 6.3.8.1, “Links for Market Data” on page 78. You can find the DBD in A.2.2, “Index Database” on page 83.

Assuming that each index is defined in only one market and that you keep history for ten days, the average database record length can be calculated as in Table 2.

Table 2. Calculating the Average Database Record Length of the Index Database

Quantity per Root	Segment Name	Data Length	Prefix Length	Segment Length	Total Length
For a local index					
1	Index	28	18	46	46
1	Market	8	8	16	16
10	History	24	8	32	320
Total					382
For a remote index					
1	Index	10	18	28	28
1	Market	8	8	16	16
Total					44

The Index segment prefix contains

- 6 bytes for the fixed part of the segment prefix
- 4 bytes for a physical child first pointer to the Market segment
- 8 bytes for a physical child first pointer and a physical child last pointer to the History segment

The Market and History segment prefixes contain

- 6 bytes for the fixed part of the segment prefix
- 2 bytes for a length field (the segment has a fixed length)

As before, we chose a small CI size (1 KB, although 0.5 KB would do just as well).

6.3.4.2 Design Considerations

This database is small and highly active, so we preload it.

6.3.5 Stock Database

This database holds details about an individual stock and a list of the stock indexes it belongs to.

6.3.5.1 Contents

If the stock trades locally, the database contains the current and historical prices of that stock as well as a log of all the deals our firm has done in that stock today. You can find the DBD in A.2.1, “Stock Database” on page 82.

We assume the following:

- Each stock is defined in only one market.
- Each stock is defined in one index. (This is likely to be an overestimate, because only important stocks belong to an index. However, the index segment is short, so the error will be small.)
- History is kept for 10 days.
- There are 160 price movements per stock per day (roughly one every 3 minutes),

As a result, the calculation is as in Table 3.

Table 3. Calculating the Average Database Record Length of the Stock Database

Quantity per Root	Segment Name	Data Length	Prefix Length	Segment Length	Total Length
For a local stock					
1	Stock	34	38	72	72
1	Index	8	8	16	16
1	Market	8	8	16	16
10	History	24	8	32	320
160	Price	12	8	20	3200
Total					3624
For a remote stock					
1	Stock	10	38	48	48
1	Market	8	8	16	16
Total					64

These aspects are considered in estimating system needs:

- We do not count the transaction log segments, because IMS stores them separately from the root.
- The average database record length depends mainly on the number of price movements per day. It will doubtless vary widely from stock to stock.
- The stock segment prefix contains
  - 6 bytes for the fixed part of the segment prefix
  - 8 bytes for an SDEP pointer
  - 4 bytes for a physical child first pointer to the index segment

- 4 bytes for a physical child first pointer to the market segment
- 8 bytes for a physical child first pointer and a physical child last pointer to the history segment
- 8 bytes for a physical child first pointer and a physical child last pointer to the price segment.

• The index, market, history and price segment prefixes contain:

- 6 bytes for the fixed part of the segment prefix
- 2 bytes for a length field (the segment has a fixed length)

We chose a CI size of 4 KB, because it can contain the expected database record.

6.3.5.2 Design Considerations

We use an SDEP segment to hold the log of deals for two reasons. First, there may be many deals, so the segments must be inserted quickly. Second, we do not need to read the deal information again until we summarize it at the close of business.

If the stock trades abroad, the database contains the market name where the details can be found. In a full-function database, this would be a logical relationship, see 6.3.8.1, “Links for Market Data” on page 78.

The stock database would benefit from its own randomizing routine. Ideally this would place stocks in an area depending on which markets they trade in. This isn′t easy with the current design because the key does not contain the market name.

Because this is a large database, so we do not preload it. It is an important database, so we chose to have two copies of each area.

6.3.6 User Database

We did not design this database. It would contain information about users, which functions they are allowed to use, and which news items they are interested in.

This database would be relatively large so we would not preload it.

6.3.7 News Database

We did not design this database. It would contain items of news, and would be a good candidate for segment compression. It might benefit from subset pointers, so that you could skip old news items. It would be a large database and we would not preload it.

6.3.8 Emulating Logical Relationships

There are relationships between the databases in the sample application.

6.3.8.1 Links for Market Data

Both the stock and index databases contain a market segment. This identifies a root on the market database or the control database. This root contains the information necessary to communicate with that market.

In a full-function application, there would be a unidirectional logical relationship to the market and control databases. IMS does not allow logical relationships for DEDBs, so we must implement the relationship ourselves. When our applications wish to follow the relationship, they must make two database calls. The first call reads the name of the market from the stock or index database and the second call reads the market details from the market or control database.

This contrasts with a full-function database, where you can retrieve the details with a single call using a logical DBD. The actual I/O is the same in both cases, so the saving is in application logic.

6.3.8.2 Link between Stocks and Indexes

This case is similar to the links for market data. The stock database contains index segments, which identify the stock indexes the stock belongs to. The application has to make two reads to retrieve the data it needs.

We could have chosen to implement a bidirectional relationship in this case, but rejected this option for two reasons:

- Only the Market Close program needs to know which stocks belong to a given index. This program reads every stock and can easily calculate the stock index values as it goes along. There are sufficiently few stock indexes to make this approach practical.
- Some indexes contain a very large number of stocks, while others contain only a few. This would lead to both long twin chains and widely varying database record lengths. Long twin chains would reduce performance in data entry.

  Widely varying database record lengths would make it difficult to chose an efficient CI size (see 5.3.2, “Picking a CI Size” on page 61 for a more detailed discussion of the impact).

6.3.9 CI Sizes in the Sample Application

We have two different CI sizes in our sample databases. The stock database has CIs of 4 KB and the other databases have CIs of 1 KB. Since there are no other databases in our sample IMS system, this means that the Fast Path database buffers must be 4 KB in size. This wastes 3 KB of buffer space for most of our databases, but this is not much of a problem because the bulk of our data is on the stock database.

This is an illustration of one of the DEDB design considerations we mentioned in 5.3.2, “Picking a CI Size” on page 61.

6.3.10 Long Twin Chains

This topic applies equally to full-function databases and DEDBs, because we did not need subset pointers in our sample application.

Our sample application has some potentially long twin chains. This topic discusses the measures we took to limit the impact these would have. During

this discussion, refer to A.2.1, “Stock Database” on page 82 and A.2.2, “Index Database” on page 83 if you want more details.

6.3.10.1 Price Segments

The stock database has price segments. There is one segment for every price change that happened today. Three programs use this segment:

- Stock Inquiry — This program sends all the price segments to the user for graphical display on the workstation. This is intended to show trends developing over time.
- Market Close — This program summarizes the price segments and then deletes them all.
- Price Notification — This program adds a price segment whenever the Stock Exchange system informs us of a price change.

We do not know how many price segments there will be, because the number depends on how active a stock is.

We decided to address these requirements in the following way.

- Storing the segments in date sequence (that is, with the oldest segments first). This makes it easier for the workstation function to display the results graphically.
- Using a physical child last pointer to speed inserts. This is the main activity on this segment, so it must be fast.

  When you insert a segment, IMS has to traverse the twin chain from the start up to the place where the segment should go. If IMS knows that a segment always goes at the end of the chain, it will use the physical child last pointer (if you supply one) to go straight to the end of the chain. However, this works only if IMS is certain that the segment will go at the end of the chain. This means that the segment must have no key and an insert rule of LAST.

  We are safe not to have a key, because the insertion sequence is also the sequence we want the segments to be stored in.

6.3.10.2 History Segments

Both the stock and index databases have history segments. These hold a summary of a day′s activity on a particular stock or index. The programs that use these segments are:

- Stock Inquiry — This program sends all the history segments to the user for graphical display on the workstation. This is intended to show trends developing over time.
- Market Close — This program creates new history segments by summarizing the day′s activity. It also deletes old history segments when they reach their retention criterion.

We do not know how many history segments there will be, because you can control the number by altering the retention criterion (the number of days to retain a segment) in the control database. In our testing, we kept only ten history segments (one for each of the last ten days), but we took steps to minimize the impact if the twin chain were long.

We decided to address these requirements in the following way.

- We store the segments in date sequence (that is, with the oldest segments first). This makes it easier for the workstation function to display the results graphically.

  It also means that the Market Close program can delete segments from the front of the twin chain. This is the efficient way to do it.

  When you go in to get a segment, IMS has to traverse the twin chain from the start to find the segment you want. If you then delete that segment, IMS has to traverse the twin chain from the start again to find the immediately preceding segment, so that it can update the twin forward pointer.[17] This becomes time consuming if the twin chain is long.

- We use a physical child last pointer to speed inserts. This is for the same as in 6.3.10.1, “Price Segments” on page 79.

[17] You can specify a twin backward pointer in a full-function database to help with this step. IMS does not allow such pointers for DEDBs.

Appendix A. Sample Application Code

As you read these examples, you will notice some features we introduced so that the London, New York, and Tokyo copies of the application could run on the same IMS system and all use the same underlying code. This process is called cloning and may appear strange if you have not encountered it before.

A.1 Cloning

Cloning is used for database, programs, definitions, and transactions.

A.1.1 Cloning PSBs and DBDs

There are three copies of every PSB and DBD, each assembled and linked separately. However, the different PSBs and DBDs are only two lines long. The first line sets a symbolic suffix that appears on all the external names and the second line copies the source itself. For example, the DBD for the London stock database looks like this:

```
&SUFF    SETC 'L'            set London suffix
         COPY STOCK          generate the Stock database
```

A.1.2 Cloning Programs

We clone the programs by creating aliases at linkage time. Because we are using the REXX interpreter, we also have to do something for the source code. We could have used IEBUPDTE to create aliases, but chose instead to use stub modules. For example, the London copy of the Stock Inquiry program looks like this:

```
         CALL STOCKI         call common process
         EXIT
```

This approach has the advantage that you can replace one of the clones with another program without affecting the others. For example, you might want to interrupt the application flow with a dummy program, or substitute the A.3.8.2, “Remove Unwanted IMS Messages” on page 124.

A.1.3 Cloning Stage One Input

We clone simple definitions manually (for example databases and BMPs), and use instream macros to generate complicated things (like transactions).

A.1.4 Cloning Transactions

It is important not to use literal transaction codes in a cloned application. The sample application always reads the transaction code from a database before doing a program switch. If one of the transactions had to reschedule itself (none do), it would use the transaction code it was originally invoked with.

A.2 Databases

This section contains annotated DBDs.

A.2.1 Stock Database

```
*
* Stock database
*
* This database holds the data relating to an individual stock.
* It is a large and  busy database.
*
* In reality we would code our own randomizer, which would assign roots
* to an area depending on business criteria. This would not be a simple
* process, because the root keys are not meaningful.
*
         DBD NAME=STOCK&SUFF,ACCESS=DEDB,RMNAME=DBFHDC44
*
* We have chosen a number of areas to minimize the impact of FP
* utilities, which operate on an area level
*
         AREA DD1=STOCK1&SUFF,SIZE=4096,UOW=(10,2),ROOT=(15,5)
         AREA DD1=STOCK2&SUFF,SIZE=4096,UOW=(10,2),ROOT=(15,5)
         AREA DD1=STOCK3&SUFF,SIZE=4096,UOW=(10,2),ROOT=(15,5)
*
* This is the root segment. It represents a single stock.
*
* NB 1.Current price is also held in the last price segment, during
*      trading hours
*    2.Stock quantity fields are limited to 6 bytes in length by the
*      IMS adapter for REXX.
*
         SEGM NAME=STOCK,PARENT=0,BYTES=(34,10)
         FIELD TYPE=C,START=03,BYTES=08,NAME=(NAME,SEQ,U)
         FIELD TYPE=P,START=11,BYTES=04,NAME=PRICE       current
         FIELD TYPE=P,START=15,BYTES=04,NAME=HIGH        highest ever
         FIELD TYPE=P,START=19,BYTES=04,NAME=LOW         lowest ever
         FIELD TYPE=P,START=23,BYTES=06,NAME=QUANTITY    held current
         FIELD TYPE=P,START=29,BYTES=06,NAME=CLOSE       holdings
*
* This segment is used to log stock transactions.
*
         SEGM NAME=TXNLOG,PARENT=STOCK,TYPE=SEQ,BYTES=(62,21)
         FIELD TYPE=C,START=03,BYTES=08,NAME=NAME        of stock
         FIELD TYPE=C,START=11,BYTES=01,NAME=ACTION      Buy or Sell
         FIELD TYPE=P,START=12,BYTES=06,NAME=QUANTITY    sold/bought
         FIELD TYPE=P,START=18,BYTES=04,NAME=PRICE
         FIELD TYPE=C,START=22,BYTES=40,NAME=DEALER      to/from whom
*
* This segment says whether this stock belongs to stock index(es)
* There are corresponding segment(s) on the INDEX database
* In a full-function environment, this would be a logical relationship.
* Although we are not sensitive to the sequence of these segments,
* they are keyed to assist the data entry and correction functions.
*
         SEGM NAME=INDEX,PARENT=STOCK,BYTES=8
         FIELD TYPE=C,START=01,BYTES=08,NAME=(NAME,SEQ,U)
*
* This segment says where this stock is quoted.
```

```
* There is a corresponding segment in the MARKET database, which says
* how to reach the market where this stock is quoted.
* If a stock is only quoted abroad, most of the root fields and
* most of the segments will not be present.
* In a full-function environment, this would be a logical relationship.
* Although we are not sensitive to the sequence of these segments,
* they are keyed to assist the data entry and correction functions.
*
         SEGM NAME=MARKET,PARENT=STOCK,BYTES=8
         FIELD TYPE=C,START=01,BYTES=08,NAME=(NAME,SEQ,U)
*
* These segments hold the last n days price data (n is held on the
* retention criteria segment of the CNTL database). They are held in
* insertion (chronological) sequence, with a Physical Child Last (PCL)
* pointer to assist the process.
*
         SEGM NAME=HISTORY,PARENT=((STOCK,DBLE)),BYTES=24,           X
              RULES=(PPP,LAST)
         FIELD TYPE=P,START=01,BYTES=05,NAME=DATE          OYYYYMMDD
         FIELD TYPE=P,START=06,BYTES=04,NAME=CLOSING       price
         FIELD TYPE=P,START=10,BYTES=04,NAME=HIGH          daily high
         FIELD TYPE=P,START=14,BYTES=04,NAME=LOW           daily low
         FIELD TYPE=P,START=18,BYTES=06,NAME=QUANTITY      traded
*
* These segments contain today's price movements. They are held in
* insertion (chronological) sequence, with a PCL pointer to speed it up.
* The Market Close program summarizes and then deletes these segments.
*
         SEGM NAME=PRICE,PARENT=((STOCK,DBLE)),BYTES=14,             X
              RULES=(PPP,LAST)
         FIELD TYPE=P,START=01,BYTES=04,NAME=TIME        hhmmsst
         FIELD TYPE=P,START=05,BYTES=04,NAME=PRICE
         FIELD TYPE=P,START=08,BYTES=06,NAME=QUANTITY    traded
*
         DBDGEN
         END
```

A.2.2 Index Database

```
*
* Index database, contains the current value of stock indexes.
* There are very few of these indexes but they are constantly updated.
* Unless we are careful, this database will be a source of contention.
*
* The sample randomizer is fine for this database because there are
* so few roots.
*
         DBD NAME=INDEX&SUFF,ACCESS=DEDB,RMNAME=DBFHDC44
*
* One area will contain the entire database.
*
         AREA DD1=INDEX1&SUFF,SIZE=1024,UOW=(2,1),ROOT=(24,4)
*
* This is the root segment. It represents a single stock index.
*
         SEGM NAME=INDEX,PARENT=0,BYTES=(28,10)
         FIELD TYPE=C,START=03,BYTES=08,NAME=(NAME,SEQ,U)
         FIELD TYPE=P,START=11,BYTES=04,NAME=VALUE       current
         FIELD TYPE=P,START=15,BYTES=04,NAME=HIVALUE     highest ever
```

```
          FIELD TYPE=P,START=19,BYTES=04,NAME=LOVALUE     lowest ever
          FIELD TYPE=P,START=23,BYTES=06,NAME=VOLUME      traded today
*
* This segment says where this index belongs.
* There is a corresponding segment in the MARKET database, which says
* how to reach the market where this index belongs.
* If an index is foreign, most of the root fields and
* the history segments will not be present.
* In a full-function environment, this would be a logical relationship
* Although we are not sensitive to the sequence of these segments,
* they are keyed to assist the data entry and correction functions.
*
          SEGM NAME=MARKET,PARENT=INDEX,BYTES=8
          FIELD TYPE=C,START=01,BYTES=08,NAME=(NAME,SEQ,U)
*
* This segment holds the last n days stock index data.
* (n is held on the retention criteria segment of the CNTL database)
* These segments are held in insertion (chronological) sequence with
* a Physical Child Last pointer to help the insert process.
*
          SEGM NAME=HISTORY,PARENT=((INDEX,DBLE)),                    X
               BYTES=24,RULES=(PPP,LAST)
          FIELD TYPE=P,START=01,BYTES=05,NAME=DATE          0YYYYMMDD
          FIELD TYPE=P,START=06,BYTES=04,NAME=CLOSING       value
          FIELD TYPE=P,START=10,BYTES=04,NAME=HIGH          records ..
          FIELD TYPE=P,START=14,BYTES=04,NAME=LOW           .. to date
          FIELD TYPE=P,START=18,BYTES=06,NAME=VOLUME        traded
*
          DBDGEN
          END
```

A.2.3 Market Database

```
*
* Market database
*
* It contains one root for every market we trade in.
*
          DBD NAME=MARKET&SUFF,ACCESS=(HDAM,VSAM),                    X
               RMNAME=(DFSHDC40,15,2)
          DATASET DD1=MARKET&SUFF
*
* This is the root segment. It represents a stock market.
*
          SEGM NAME=MARKET,PARENT=0,BYTES=60
          FIELD TYPE=C,START=01,BYTES=08,NAME=(NAME,SEQ,U)
          FIELD TYPE=C,START=09,BYTES=03,NAME=CURRENCY
          FIELD TYPE=C,START=12,BYTES=01,NAME=OPEN        Yes or No
*
* APPC info on how to route to our office in this market
* (blank for the local market).
*
          FIELD TYPE=C,START=13,BYTES=08,NAME=SIDEINFO    for APPC/MVS
          FIELD TYPE=C,START=21,BYTES=08,NAME=LOGMODE     for VTAM
          FIELD TYPE=C,START=29,BYTES=08,NAME=PARTNER     LU name
*
* Transaction codes belonging to this office
*
          FIELD TYPE=C,START=37,BYTES=08,NAME=STOCKI  stock inquiry
```

```
          FIELD TYPE=C,START=45,BYTES=08,NAME=STATUS  open/close notify
          FIELD TYPE=C,START=53,BYTES=08,NAME=TRADE   for future use
*
          DBDGEN
          END
```

A.2.4 Currency Database

```
*
* Currency database, contains one root for each foreign office       .
*
*
* The sample randomizer will do, because this is a very small database
*
          DBD NAME=CURR&SUFF,ACCESS=DEDB,RMNAME=DBFHDC44
*
* This is a very small databse and won't easily split into areas
*
          AREA DD1=CURR1&SUFF,SIZE=1024,UOW=(2,1),ROOT=(5,2)
*
* This is the root segment. It represents a currency
*
          SEGM NAME=CURRENCY,PARENT=0,BYTES=10
          FIELD TYPE=C,START=01,BYTES=03,NAME=(NAME,SEQ,U)
          FIELD TYPE=C,START=04,BYTES=01,NAME=TYPE        * or /
          FIELD TYPE=P,START=05,BYTES=06,NAME=RATE       (7)9.9999
*
          DBDGEN
          END
```

A.2.5 Control Database

```
*
* Control database, contains reference information                   .
*
          DBD NAME=CNTL&SUFF,ACCESS=(HDAM,VSAM),RMNAME=(DFSHDC40,5,2)
          DATASET DD1=CNTL&SUFF
*
* This is the root segment ........
*
          SEGM NAME=CONTROL,PARENT=0,BYTES=(38,8)
          FIELD TYPE=C,START=03,BYTES=02,NAME=(TYPE,SEQ,U)
*
* ...... and there are several kinds:
*
* Local Market Name segment (type = 'LO')
* Start Length Type Contents
*   5     8      C  Name of local market
*
*
* Data Retention Criteria (type = 'R1')
* Start Length Type Contents
*   5     2      P  Number of days history to hold on Stock DB
*   7     2      P  Number of days history to hold on Index DB
*
*
* Local Stock exchange system segment (type = 'SE')
* Start Length Type Contents
*   5     8      C  TPname of Local Stock Exchage System
*  13     8      C  APPC/MVS side info for ditto
```

```
*  21     8      C  VTAM log mode for ditto
*  29     8      C  VTAM LU name for ditto
*
          DBDGEN
          END
```

A.2.6 DBRC Definitions

This is a subset of the DBRC definitions used for testing. We show the databases for the London copy of the application. DBRC does not support symbolic variables, so we were forced to create three copies of this input.

You will notice that the stock database uses MADS.

```
 /*                                                                 */
 /* Control database - London                                       */
 /*                                                                 */

 INIT.DB    DBD(CNTLL) SHARELVL(1) TYPEIMS

 INIT.DBDS  DBD(CNTLL) DDN(CNTLL) ICJCL(ICJCL) RECOVJCL(RECOVJCL) -
            REUSE RECOVPD(0) GENMAX(3) DSN(IMS510.STOCK.CNTLL)

 INIT.IC    .... (details of three image copies) ....

 /*                                                                 */
 /* Currency database - London                                      */
 /*                                                                 */

 INIT.DB    DBD(CURRL) SHARELVL(1) TYPEFP

 INIT.DBDS  DBD(CURRL) AREA(CURR1L) ICJCL(ICJCL) RECOVJCL(RECOVJCL) -
            REUSE RECOVPD(0) GENMAX(3) VSO PRELOAD PREOPEN

 INIT.ADS   DBD(CURRL) AREA(CURR1L) -
            ADDN(CURR1L) ADSN(IMS510.STOCK.CURR1L) UNAVAIL

 INIT.IC    .... (details of three image copies) ....

 /*                                                                 */
 /* Stock Index database - London                                   */
 /*                                                                 */

 INIT.DB    DBD(INDEXL) SHARELVL(1) TYPEFP

 INIT.DBDS  DBD(INDEXL) AREA(INDEX1L) ICJCL(ICJCL) RECOVJCL(RECOVJCL) -
            REUSE(RECOVPD(0) GENMAX(3) VSO PRELOAD PREOPEN

 INIT.ADS   DBD(INDEXL) AREA(INDEX1L)
            ADDN(INDEX1L) ADSN(IMS510.STOCK.INDEX1L) UNAVAIL

 INIT.IC    .... (details of three image copies) ....

 /*                                                                 */
 /* Market database - London                                        */
 /*                                                                 */

 INIT.DB    DBD(MARKETL) SHARELVL(1) TYPEIMS

 INIT.DBDS  DBD(MARKETL) DDN(MARKETL) ICJCL(ICJCL) RECOVJCL(RECOVJCL) -
```

```
           REUSE RECOVPD(0) GENMAX(3) DSN(IMS510.STOCK.MARKETL)

INIT.IC    .... (details of three image copies) ....

/*                                                                  */
/* Stock database - London                                          */
/* The Stock database has three areas, each of which has two copies.*/
/*                                                                  */

INIT.DB    DBD(STOCKL) SHARELVL(1) TYPEFP

INIT.DBDS  DBD(STOCKL) AREA(STOCK1L) ICJCL(ICJCL) RECOVJCL(RECOVJCL) -
           REUSE RECOVPD(0) GENMAX(3) VSO PREOPEN

INIT.ADS   DBD(STOCKL) AREA(STOCK1L) -
           ADDN(STOCK1L1) ADSN(IMS510.STOCK.STOCK1L1) UNAVAIL

INIT.ADS   DBD(STOCKL) AREA(STOCK1L) -
           ADDN(STOCK1L2) ADSN(IMS510.STOCK.STOCK1L2) UNAVAIL

INIT.IC    .... (details of three image copies) ....

INIT.DBDS  DBD(STOCKL) AREA(STOCK2L) ICJCL(ICJCL) RECOVJCL(RECOVJCL) -
           REUSE RECOVPD(0) GENMAX(3) VSO PREOPEN

INIT.ADS   DBD(STOCKL) AREA(STOCK2L) -
           ADDN(STOCK2L1) ADSN(IMS510.STOCK.STOCK2L1) UNAVAIL

INIT.ADS   DBD(STOCKL) AREA(STOCK2L) -
           ADDN(STOCK2L2) ADSN(IMS510.STOCK.STOCK2L2) UNAVAIL

INIT.IC    .... (details of three image copies) ....

INIT.DBDS  DBD(STOCKL) AREA(STOCK3L) ICJCL(ICJCL) RECOVJCL(RECOVJCL) -
           REUSE RECOVPD(0) GENMAX(3) VSO PREOPEN

INIT.ADS   DBD(STOCKL) AREA(STOCK3L) -
           ADDN(STOCK3L1) ADSN(IMS510.STOCK.STOCK3L1) UNAVAIL

INIT.ADS   DBD(STOCKL) AREA(STOCK3L) -
           ADDN(STOCK3L2) ADSN(IMS510.STOCK.STOCK3L2) UNAVAIL

INIT.IC    .... (details of three image copies) ....
```

A.3 Programs

This section contains REXX program source and PSBs. The programs are rather short and intolerant of unexpected conditions (&eg, a database segment being missing when it should be found). We did this deliberately for two reasons; so that the examples would be short and uncluttered, and so that we could create more examples in the time available.

A.3.1 Stock Inquiry Functions

This function gathers stock information from various sources.

A.3.1.1 Stock Inquiry PSB

```
*
* Stock Inquiry MPP
*
          PUNCH ' ALIAS STOCKI&SUFF.F '
STOCK     PCB TYPE=DB,NAME=STOCK&SUFF,PROCOPT=G,KEYLEN=16
          SENSEG NAME=STOCK,PARENT=0
          SENSEG NAME=INDEX,PARENT=STOCK
          SENSEG NAME=MARKET,PARENT=STOCK
          SENSEG NAME=HISTORY,PARENT=STOCK
          SENSEG NAME=PRICE,PARENT=STOCK
INDEX     PCB TYPE=DB,NAME=INDEX&SUFF,PROCOPT=G,KEYLEN=16
          SENSEG NAME=INDEX,PARENT=0
          SENSEG NAME=MARKET,PARENT=INDEX
          SENSEG NAME=HISTORY,PARENT=INDEX
CONTROL   PCB TYPE=DB,NAME=CNTL&SUFF,PROCOPT=GOT,KEYLEN=2
          SENSEG NAME=CONTROL,PARENT=0
MARKET    PCB TYPE=DB,NAME=MARKET&SUFF,PROCOPT=GOT,KEYLEN=8
          SENSEG NAME=MARKET,PARENT=0
CURRENCY  PCB TYPE=DB,NAME=CURR&SUFF,PROCOPT=G,KEYLEN=3
          SENSEG NAME=CURRENCY,PARENT=0
*
          PSBGEN PSBNAME=STOCKI&SUFF,LANG=ASSEM,CMPAT=YES
          END
```

A.3.1.2 Stock Inquiry Program for IMS

This program uses A.3.1.3, “APPC Driver” on page 94.

```
/* Stock Inquiry - IMS                                                */
/*                                                                    */
/* This transaction is invoked by a user on a PWS who wishes to know */
/* more about a certain stock. There are five categories of data the */
/* user can request:                                                  */
/*   Current price                                                    */
/*   Price movements today                                            */
/*   Historical price information                                     */
/*   Current Stock Index                                              */
/*   Historical Stock Index information                               */
/*                                                                    */
/* Refer to PWS stock inquiry program for message layouts.            */
/*                                                                    */
/* This information might not be held on the local system, so this    */
/* tranaction is able to converse with other copies of itself in      */
/* other offices.                                                     */
/*                                                                    */
/* We implemented this transaction using explicit LU 6.2 calls, so   */
/* we can retrieve data from abroad synchronously.                    */
/*                                                                    */
/* Logic                                                              */
/*                                                                    */
/* Get a message                                                      */
/* Determine local market ('LO' segment on Control DB)                */
/* For each message                                                   */
/*   Process for Stock inquiries                                      */
/*     Get stock DB root                                              */
```

```
/*     Read every market this stock is traded in                    */
/*     For each remote market ...                                   */
/*       Initiate conversation                                      */
/*       Send request                                               */
/*       Wait for response                                          */
/*       Read currency database and convert monetary values         */
/*       Terminate conversation                                     */
/*       Add requested details to reply for user                    */
/*     For local market ...                                         */
/*       Add requested details to reply for user                    */
/*   Process for Stock Index inquiries                              */
/*     Get index DB root                                            */
/*     Read every market this index exists on (index DB + market DB) */
/*     For each remote market ...                                   */
/*       Initiate conversation                                      */
/*       Send request                                               */
/*       Wait for response                                          */
/*       Terminate conversation                                     */
/*       Add requested details to reply for user                    */
/*     For local market ...                                         */
/*       Add requested details to reply for user                    */
/*   Send reply to user                                             */
/*   Get next message                                               */
/*                                                                  */
ADDRESS REXXIMS
'IMSRXTRC 0'
/*                                                                  */
/* Layout of local market segment on Control DB                     */
/*                                                                  */
MAP = '.  C 4 : LOCAL_MARKET C 8'
'MAPDEF LO_SEG MAP REPLACE'
/*                                                                  */
/* Stock segment                                                    */
/*                                                                  */
MAP = '.  C 10 : PRICE P 4 : . C 8 : QTY P 6 : . P 6 : '
'MAPDEF STOCK_SEG MAP REPLACE'
/*                                                                  */
/* Index segment                                                    */
/*                                                                  */
MAP = '.  C 10 : VAL P 4 : . C 14'
'MAPDEF INDEX_SEG MAP REPLACE'
/*                                                                  */
/* Program constants                                                */
/*                                                                  */
ERR_LAB = 'Error-'
/*                                                                  */
/* Determine the local market                                       */
/*                                                                  */
SSA = 'CONTROL (TYPE    = LO)'
'GU CONTROL *LO_SEG SSA'
/*                                                                  */
/* Get message                                                      */
/*                                                                  */
'GU IOPCB INMSG'
DO WHILE IMSQUERY('STATUS') = '  '
   /*                                                               */
   /* Clear out any rubbish from a previous iteration               */
   /*                                                               */
   RESPONSE = ''
```

```
/*                                                                  */
/* Decode input message                                             */
/*                                                                  */
PARSE UPPER VAR INMSG . REQUEST NAME
NAME    = SUBSTR(NAME,1,8,' ')   /* pad to full length for SSAs   */
SELECT
   /*                                                               */
   /* Process for Stock Inquiries                                   */
   /*                                                               */
   WHEN (REQUEST = 'SC' | REQUEST = 'SD' | ,
         REQUEST = 'SH' | REQUEST = 'SQ') THEN DO
      /*                                                            */
      /* Get Stock Root and first market                            */
      /*                                                            */
      SSA1 = 'STOCK   *P(NAME    = ' || NAME || ')'
      SSA2 = 'MARKET   '
      'GU   STOCK *STOCK_SEG  SSA1'
      IF IMSQUERY('STATUS') = '  ' THEN DO
         I = 0
         'GNP  STOCK MARKET_NAME SSA2'
         /*                                                         */
         /* Process every market this stock trades in               */
         /*                                                         */
         DO WHILE (IMSQUERY('STATUS') = '  ')
            I = I + 1
            /*                                                      */
            /* Determine if market is local or remote              */
            /*                                                      */
            IF MARKET_NAME = LOCAL_MARKET
            THEN CALL GET_LOCAL_STOCK  LOCAL_MARKET REQUEST NAME,
                                                    PRICE QTY
            ELSE CALL GET_REMOTE       MARKET_NAME  REQUEST NAME
            /*                                                      */
            /* Get next market                                      */
            /*                                                      */
            'GNP  STOCK MARKET_NAME SSA2'
         END
         IF I = 0
         THEN RESPONSE = ERR_LAB NAME 'is not traded on any market'
      END
      ELSE RESPONSE = ERR_LAB NAME 'is not a known stock'
   END
   /*                                                               */
   /* Process for Index Inquiries                                   */
   /*                                                               */
   WHEN (REQUEST = 'IC' | REQUEST = 'IH') THEN DO
      /*                                                            */
      /* Get Index root and first market                            */
      /*                                                            */
      SSA1 = 'INDEX   *P(NAME    = ' || NAME || ')'
      SSA2 = 'MARKET   '
      'GU   INDEX *INDEX_SEG   SSA1'
      IF IMSQUERY('STATUS') = '  ' THEN DO
         I = 0
         'GNP  INDEX MARKET_NAME  SSA2'
         /*                                                         */
         /* Process every market                                    */
         /*                                                         */
         DO WHILE  IMSQUERY('STATUS') = '  '
```

```
            I = I + 1
            /*                                                    */
            /* Determine if market is local or remote             */
            /*                                                    */
            IF MARKET_NAME = LOCAL_MARKET
            THEN CALL GET_LOCAL_INDEX  LOCAL_MARKET REQUEST NAME VAL
            ELSE CALL GET_REMOTE       MARKET_NAME  REQUEST NAME
            /*                                                    */
            /* Get next market                                    */
            /*                                                    */
            'GNP  INDEX MARKET_NAME SSA2'
         END
         IF I = 0
         THEN RESPONSE = ERR_LAB NAME 'is not defined in any market'
      END
      ELSE RESPONSE = ERR_LAB NAME 'is not a known index'
   END
   OTHERWISE
      RESPONSE = ERR_LAB REQUEST 'is not a recognized request code'
END
/*                                                                */
/* Send the response                                              */
/*                                                                */
'ISRT IOPCB RESPONSE'
/*                                                                */
/* Get next message                                               */
/*                                                                */
'GU   IOPCB INMSG'
END
EXIT
/*                                                                */
/* Get data from a remote market                                  */
/*                                                                */
GET_REMOTE: PROCEDURE EXPOSE RESPONSE ERR_LAB
   PARSE ARG MARKET_NAME REQUEST NAME
   /*                                                             */
   /* Market segment (from Market DB)                            */
   /*                                                             */
   MAP = '.       C 8 : CURR    C 3 : OPEN   C 1 : SIDEINFO C 8 :',
         'LOGMODE C 8 : PARTNER C 8 : STOCKI C 8'
   'MAPDEF MARKET_SEG MAP REPLACE'
   /*                                                             */
   /* Currency segment (from Currency DB)                        */
   /*                                                             */
   MAP = '.  C 3 : TYPE C 1 : RATE P 6'
   'MAPDEF CURR_SEG MAP REPLACE'
   /*                                                             */
   /* Get market segment for APPC parameters                     */
   /*                                                             */
   SSA = 'MARKET  (NAME    = ' || SUBSTR(MARKET_NAME,1,8,' ') || ')'
   'GU MARKET *MARKET_SEG SSA'
   IF IMSQUERY('STATUS') \= '  ' THEN EXIT
   /*                                                             */
   /* Do the APPC bits                                           */
   /*                                                             */
   OUTMSG   = REQUEST NAME
   RESPONSE = APPCDRV(PARTNER,LOGMODE,SIDEINFO,STOCKI,OUTMSG)
   /*                                                             */
   /* Inspect reply for error information                        */
```

```
   /*                                                                */
   PARSE VAR RESPONSE MKT RESPONSE
   /*                                                                */
   /* Messages starting withe ERR_LAB are errors                     */
   /* Request types IC and IH do not require currency conversion     */
   /* neither should be processed further                            */
   /*                                                                */
   /*                                                                */
   IF MKT = ERR_LAB | REQUEST = 'IH' | REQUEST = 'IC'
   THEN RESPONSE = MKT RESPONSE;
   /*                                                                */
   /* Convert currency amounts as needed                             */
   /*                                                                */
   ELSE DO
      /*                                                             */
      /* Get currency segment; we will need it                       */
      /*                                                             */
      SSA = 'CURRENCY(NAME    = ' || SUBSTR(CURR,1,3,' ') || ')'
      'GU CURRENCY *CURR_SEG SSA'
      SELECT
         WHEN (REQUEST = 'SC' | REQUEST = 'SQ') THEN DO
            PARSE VAR RESPONSE VAL .
            RESPONSE = MKT CONVERT(VAL,TYPE,RATE)
         END
         WHEN (REQUEST = 'SD') THEN DO
            PARSE VAR RESPONSE TIM VAL RESPONSE
            RESPONSE2 = MKT
            DO UNTIL RESPONSE = ''
               RESPONSE2 = RESPONSE2 TIM CONVERT(VAL,TYPE,RATE)
               PARSE VAR RESPONSE TIM VAL RESPONSE
            END
            RESPONSE = RESPONSE2
         END
         WHEN (REQUEST = 'SH') THEN DO
            PARSE VAR RESPONSE DAT CLOS HI LOW VOL RESPONSE
            RESPONSE2 = MKT
            DO UNTIL RESPONSE = ''
               RESPONSE2 = RESPONSE2 DAT CONVERT(CLOS,TYPE,RATE)
                           CONVERT(HI,TYPE,RATE) ,
                           CONVERT(LOW,TYPE,RATE) VOL
               PARSE VAR RESPONSE DAT CLOS HI LOW VOL RESPONSE
            END
            RESPONSE = RESPONSE2
         END
      END
   END
RETURN
/*                                                                */
/* Convert from one currency to another                          */
/*                                                                */
CONVERT: PROCEDURE
   PARSE ARG VAL, TYPE, RATE
   IF TYPE = '*'
   THEN RETURN TRUNC( VAL * RATE / 10000)
   ELSE RETURN TRUNC( VAL / RATE * 10000)
/*                                                                */
/* Get data about a local stock                                   */
/*                                                                */
GET_LOCAL_STOCK: PROCEDURE EXPOSE RESPONSE ERR_LAB
```

```
   PARSE ARG LOCAL_MARKET REQUEST NAME PRICE QTY
   /*                                                                */
   /* Decode request type                                            */
   /*                                                                */
   SELECT
      /*                                                            */
      /* Process a request for current price info                   */
      /*                                                            */
      WHEN (REQUEST = 'SC') THEN
         RESPONSE = LOCAL_MARKET PRICE
      /*                                                            */
      /* Process a request for current holdings                     */
      /*                                                            */
      WHEN (REQUEST = 'SQ') THEN
         RESPONSE = LOCAL_MARKET QTY
      /*                                                            */
      /* Process a request for today's price movements              */
      /*                                                            */
      WHEN (REQUEST = 'SD') THEN DO
         /*                                                         */
         /* Layout of the price segment                             */
         /*                                                         */
         MAP   = 'TIM P 4 : PRICE P 4'
         'MAPDEF PRICE MAP REPLACE'
         /*                                                         */
         RESPONSE = LOCAL_MARKET
         SSA   = 'PRICE    '
         I = 0
         'GNP STOCK *PRICE SSA'
          DO WHILE IMSQUERY('STATUS') = '  '
             I = I + 1
             RESPONSE = RESPONSE TIM PRICE
           'GNP STOCK *PRICE SSA'
          END
          IF I = 0
          THEN RESPONSE = ERR_LAB 'No price information available'
      END
      /*                                                            */
      /* Process a request for historical price movements           */
      /*                                                            */
      WHEN (REQUEST = 'SH') THEN
         CALL HISTORY STOCK
   END
RETURN
/*                                                                  */
/* Get data about a local index                                     */
/*                                                                  */
GET_LOCAL_INDEX: PROCEDURE EXPOSE RESPONSE ERR_LAB
   PARSE ARG LOCAL_MARKET REQUEST NAME VAL
   /*                                                               */
   /* Decode request type                                           */
   /*                                                               */
   SELECT
      /*                                                           */
      /* Process a request for current price info                  */
      /*                                                           */
      WHEN (REQUEST = 'IC') THEN
         RESPONSE = LOCAL_MARKET VAL
      /*                                                           */
```

```
      /* Process a request for historical price movements           */
      /*                                                            */
      WHEN (REQUEST = 'IH') THEN
         CALL HISTORY INDEX
   END
RETURN
/*                                                                  */
/* Process the history segments on a database                       */
/*                                                                  */
HISTORY: PROCEDURE  EXPOSE RESPONSE LOCAL_MARKET ERR_LAB
PARSE ARG DATABASE
   /*                                                               */
   /* Layout of the history segment                                */
   /*                                                               */
   MAP   = 'DAT P 5 : CLOSING P 4 : HIGH P 4 : LOW P 4 : QTY P 6'
   'MAPDEF HISTORY MAP REPLACE'
   /*                                                               */
   RESPONSE = LOCAL_MARKET
   I = 0
   SSA   = 'HISTORY  '
   'GNP DATABASE *HISTORY SSA'
   DO WHILE IMSQUERY('STATUS') = '  '
      I = I + 1
      RESPONSE = RESPONSE DAT CLOSING HIGH LOW QTY
      'GNP DATABASE *HISTORY SSA'
   END
   IF I = 0 THEN RESPONSE = ERR_LAB 'No history information available'
RETURN
```

A.3.1.3 APPC Driver

This subroutine contains the CPIC calls to drive one of our IMS transactions.

```
/* REXX */
/*                                                                  */
/* CPIC routine to drive an IMS transaction synchronously.          */
/*      it works the same on TSO, IMS or  OS/2                      */
/*                                                                  */
/* Usage:                                                           */
/*        CALL APPCDRV(luname,     VTAM LU name or blank            */
/*                     logmode,    VTAM log mode or blank           */
/*                     tp name,    transaction code                 */
/*                     sideinfo,   side information (if used)       */
/*                     message,    message to send (note 1)         */
/*                     'TRACE');   specify this parm for diagnostics */
/* Notes:                                                           */
/*   (1) Remember that IMS will place the transaction code and a    */
/*       blank in front of this message.                            */
/*                                                                  */
PARSE ARG LUNAME, LOGMODE, SIDEINFO, TPN, OUTMSG, TRACE
IF TRACE = 'TRACE' THEN
SAY       1 LUNAME 2 LOGMODE 3 SIDEINFO 4 TPN 5 OUTMSG
LUNAME   = STRIP(LUNAME,  BOTH)
LOGMODE  = STRIP(LOGMODE, BOTH)
SIDEINFO = STRIP(SIDEINFO,BOTH)
TPN      = STRIP(TPN,     BOTH)
/*                                                                  */
/* Initiate conversation (note: SIDEINFO may be blank).             */
/*                                                                  */
CALL ON ERROR NAME CPIC_FAIL
```

```
ADDRESS CPICOMM
'CMINIT CONV SIDEINFO RC'
IF TRACE = 'TRACE' THEN
SAY 'Initialize  ' "'"SIDEINFO"'" C2X(CONV) RC
/*                                                                  */
/* Set VTAM LU name (if specified)                                  */
/*                                                                  */
L = LENGTH(LUNAME)
IF L > 0 THEN DO
  'CMSPLN CONV LUNAME   L  RC'
   IF TRACE = 'TRACE' THEN
   SAY 'Set LU name ' "'"LUNAME"'" L RC
END
/*                                                                  */
/* Set VTAM log mode (if specified)                                 */
/*                                                                  */
L = LENGTH(LOGMODE)
IF L > 0 THEN DO
   'CMSMN  CONV LOGMODE  L  RC'
   IF TRACE = 'TRACE' THEN
   SAY 'Set log mode' "'"LOGMODE"'" L RC
END
/*                                                                  */
/* Set TP name (if specified) remembering to drop trailing blanks   */
/*                                                                  */
L = LENGTH(TPN)
IF L > 0 THEN DO
   'CMSTPN CONV TPN    L  RC'
   IF TRACE = 'TRACE' THEN
   SAY 'Set TP name ' "'"TPN"'" L RC
END
/*                                                                  */
/* Set sync level confirm                                           */
/*                                                                  */
'CMSSL CONV 1 RC '      /* 0 = none, 1 = confirm */
IF TRACE = 'TRACE' THEN
SAY 'Set sync lev' RC
/*                                                                  */
/* Allocate the conversation                                        */
/*                                                                  */
'CMALLC CONV RC '
IF TRACE = 'TRACE' THEN
SAY 'Allocate    ' RC
/*                                                                  */
/* Send request to IMS                                              */
/*                                                                  */
L = LENGTH(OUTMSG)
'CMSEND CONV OUTMSG L RTS RC'
IF TRACE = 'TRACE' THEN
SAY 'Send        ' "'"OUTMSG"'" L RC
/*                                                                  */
/* Await confirmation from IMS                                      */
/*                                                                  */
RESPONSE = ''
'CMRCV  CONV APPCMSG 1000 DR LEN SR RTS RC'
IF TRACE = 'TRACE' THEN
SAY 'Receive     ' DR SR LEN RC
DO I = 1 TO 10 WHILE RC = 0
   SELECT
```

```
      /*                                                              */
      /* If no data was returned, try for more                        */
      /*                                                              */
      WHEN (DR = 0) THEN DO                  /* no data returned     */
         'CMRCV  CONV APPCMSG 1000 DR LEN SR RTS RC'
         IF TRACE = 'TRACE' THEN
         SAY 'Receive     ' DR SR LEN RC
      END
      /*                                                              */
      /* If some data was returned, add it to the response and        */
      /* ask for more data.                                            */
      /*                                                              */
      WHEN (DR = 1 | DR = 2) THEN DO         /* data or partial data */
         IF LEN > 0 THEN DO
            RESPONSE = RESPONSE SUBSTR(APPCMSG,1,LEN)
            IF TRACE = 'TRACE' THEN
            SAY SUBSTR(APPCMSG,1,LEN)
         END
         'CMRCV  CONV APPCMSG 1000 DR LEN SR RTS RC'
         IF TRACE = 'TRACE' THEN
         SAY 'Receive     ' DR SR LEN RC
      END
      /*                                                              */
      /* If complete data was returned, add it to the response        */
      /*                                                              */
      WHEN (DR = 3) THEN                     /* complete data        */
         IF LEN > 0 THEN DO
            RESPONSE = RESPONSE SUBSTR(APPCMSG,1,LEN)
            IF TRACE = 'TRACE' THEN
            SAY SUBSTR(APPCMSG,1,LEN)
            LEAVE
         END
      OTHERWISE NOP                          /* no other values      */
   END
   /*                                                                 */
   /* If IMS asked for confirmation, supply it.                       */
   /*                                                                 */
   IF SR = 2 THEN DO
      'CMCFMD CONV RC'
       IF TRACE = 'TRACE' THEN
       SAY 'Confirmed   ' RC
       LEAVE
   END
END
RETURN RESPONSE
/*                                                                    */
/* Subroutine to trap CPIC errors, even if trace is off               */
/*                                                                    */
CPIC_FAIL: PROCEDURE EXPOSE RC
SAY CONDITION('D') RC
EXIT
```

A.3.1.4 Stock Inquiry Program for Workstation

This program uses the following subroutines:

- A.3.1.5, “Stock Inquiry Parameter Prompter” on page 97.
- A.3.1.6, “APPC Parameter Read for Workstation” on page 98.

```
/* REXX */
/* Stock Inquiry - PWS                                              */
/*                                                                  */
/* This transaction drives the IMS Stock Inquiry transaction, which */
/* returns data about stocks and stock indices.                     */
/*                                                                  */
/* Logic                                                            */
/*                                                                  */
/* Get request(s) from the user                                     */
/* Drive the IMS transaction                                        */
/* Show the replies to the user                                     */
/*  - this involves a graphical display of the data                 */
/*                                                                  */
/* Parse user's request                                             */
/*                                                                  */
PARSE ARG REQUEST NAME
OUTMSG   = STOCKICM(REQUEST, NAME)
/*                                                                  */
/* Do the APPC process                                              */
/*                                                                  */
RESPONSE = APPCPC('STOCKI', OUTMSG)
SAY RESPONSE
/*                                                                  */
/* This is where the fancy graphics would be done                   */
/*                                                                  */
EXIT
```

A.3.1.5 Stock Inquiry Parameter Prompter

This subroutine prompts the user for parameters needed by Stock Inquiry.

```
/* REXX */
/* Stock Inquiry - Common process                                   */
/*                                                                  */
/* This function decodes the users request and turns it into a    h */
/* message for IMS. This is the same on TSO and on the PWS.         */
/*                                                                  */
/* The user can request:                                            */
/*   SC - Current stock price                                       */
/*   SQ - Current stock holding                                     */
/*   SD - Price movements today                                     */
/*   SH - Historical price information                              */
/*   IC - Current Stock Index                                       */
/*   IH - Historical Stock Index information                        */
/*                                                                  */
/*                                                                  */
/* Interface Layout (requests to IMS)                               */
/*       1.Request code (from above table)                          */
/*       2.Name of stock or index                                   */
/*                                                                  */
/* Interface Layout (replies from IMS)                              */
/*   The response layout depends on the request type:               */
/*                                                                  */
/* Current Stock Price, Current Stock Holding or Current Stock Index */
/*   A number of elements, each with the following layout:          */
/*       1.Market where data was found                              */
/*       2.Current value                                            */
/*                                                                  */
/* Daily Stock Price Movements                                      */
/*   A number of elements, each with the following layout:          */
```

```
/*       1.Market where data was found                                */
/*     2*n.Time (hhmmsst)                                             */
/*   2*n+1.Stock price                                                */
/*                       where n is the number of price changes today */
/*                                                                    */
/* Historical Stock Price or Historical Stock Index                   */
/*   A number of elements, each with the following layout:            */
/*       1.Market where data was found                                */
/*     5*n.Date YYYYMMDD                                              */
/*   5*n+1.Closing value                                              */
/*   5*n+2.Daily High                                                 */
/*   5*n+3.Daily Low                                                  */
/*   5*n+4.Volume of shares traded                                    */
/*                         where n is the number of days history kept */
/*                                                                    */
/* Note: The IMS transaction converts stock prices to local currency,*/
/*       but leaves index values unchanged.                           */
/*                                                                    */
/* Logic                                                              */
/*                                                                    */
/* Get request(s) from the user                                       */
/* Format the message for IMS                                         */
/*                                                                    */
/* Parse user's request                                               */
/*                                                                    */
PARSE ARG REQUEST, NAME
/*                                                                    */
IF REQUEST = '' THEN DO
   SAY "Request code?"
   PULL REQUEST
END
REQUEST = TRANSLATE(REQUEST)              /* converts to upper case */
/*                                                                    */
IF NAME  = '' THEN DO
   SAY "Name of stock or Index?"
   PULL NAME
END
/*                                                                    */
/* Get APPC parameters from file                                      */
/*                                                                    */
RETURN REQUEST NAME
```

A.3.1.6 APPC Parameter Read for Workstation

This program uses the following subroutines:

- A.3.1.3, "APPC Driver" on page 94.
- A.3.1.7, "Conversion between ASCII and EBCDIC" on page 99.

```
/*                                                                    */
/* Subroutine to sens an APPC message to an IMS transaction and       */
/* elicit a response.                                                 */
/*                                                                    */
/* Usage:                                                             */
/*                                                                    */
/*       RESPONSE = APPCPC ('STOCKI' , message)                       */
/*       RESPONSE = APPCPC ('CDEAL'  , message)                       */
/*                                                                    */
PARSE ARG TRANNAME , OUTMSG
/*                                                                    */
/* Get APPC parameters from file                                      */
```

```
/*                                                                  */
FILENAME = 'LOCAL.DATA'
STATUS   = STREAM(FILENAME, 'C', 'OPEN READ')
IF STATUS \= 'READY:' THEN DO; SAY STATUS; EXIT ;END
/*                                                                  */
/* Process until end of file, or non-comment found                  */
/*                                                                  */
DO WHILE LINES(FILENAME) > 0
   INPARMS = LINEIN(FILENAME)
   IF SUBSTR(INPARMS,1,1) \= '*' THEN LEAVE
END
STATUS   = STREAM(FILENAME, 'C', 'CLOSE')              /* close file */
/*                                                                  */
/* Decode LU 6.2 parameters, dropping blanks where necessary        */
/*                                                                  */
LUNAME   = SUBSTR(INPARMS,01,17) /* IMS's LU name                   */
LOGMODE  = SUBSTR(INPARMS,18,08) /* VTAM logmode                    */
SIDEINFO = SUBSTR(INPARMS,26,08) /* APPC side info                  */
STOCKI   = SUBSTR(INPARMS,34,08) /* txn for Stock Inquiry           */
CDEAL    = SUBSTR(INPARMS,42,08) /* txn for Deal Notify             */
/*                                                                  */
/* Select the correct transaction code                              */
/*                                                                  */
SELECT
  WHEN (TRANNAME) = 'STOCKI' THEN TPN = STOCKI
  WHEN (TRANNAME) = 'CDEAL'  THEN TPN = CDEAL
  OTHERWISE EXIT
END
/*                                                                  */
/* Call the APPC driver                                             */
/*                                                                  */
OUTMSG = TRANSLATE(OUTMSG, A2E())      /* translate ASCII  to EBCDIC */
OUTMSG = APPCDRV(LUNAME,LOGMODE,SIDEINFO,TPN,OUTMSG) /* send + rcv   */
RETURN   TRANSLATE(OUTMSG, E2A())      /* translate EBCDIC to ASCII  */
```

A.3.1.7 Conversion between ASCII and EBCDIC

One of the problems with using a workstation is converting between ASCII and EBCDIC. We chose to address this problem with a pair of REXX functions, which return a 256 byte character string you can use with the TRANSLATE function.

EBCDIC to ASCII

```
/*                                                                  */
/* Function to return a translation string for use in conversion    */
/* from EBCDIC to ASCII.                                            */
/*                                                                  */
/* Usage:                                                           */
/*        ASCII = E2A()                                             */
/*        x  =  TRANSLATE(ebcdic string,ASCII)                      */
/*                                                                  */
/*        character                                                 */
/*        EBCDIC    '000102030405060708090A0B0C0D0E0F'X             */
 ASCII  =           '000102030405060708090A0B0C0D0E0F'X
/*                                                                  */
/*        character                                                 */
/*        EBCDIC    '101112131415161718191A1B1C1D1E1F'X             */
 ASCII  = ASCII  || '101112131415161718191A1B1C1D1E1F'X
/*                                                                  */
```

```
/*       character                                                  */
/*       EBCDIC    '202122232425262728292A2B2C2D2E2F'X              */
 ASCII  = ASCII   || '202122232425262728292A2B2C2D2E2F'X
/*                                                                  */
/*       character                                                  */
/*       EBCDIC    '303132333435363738393A3B3C3D3E3F'X              */
 ASCII  = ASCII   || '303132333435363738393A3B3C3D3E3F'X
/*                                                                  */
/*       character   b                       . < ( +                */
/*       EBCDIC    '404142434445464748494A4B4C4D4E4F'X              */
 ASCII  = ASCII   || '204142434445464748494A2E3C282B4F'X
/*                                                                  */
/*       character   &                     ! $ * ) ;                */
/*       EBCDIC    '505152535455565758595A5B5C5D5E5F'X              */
 ASCII  = ASCII   || '265152535455565758592124 2A293B5F'X
/*                                                                  */
/*       character   - /                     , %   > ?              */
/*       EBCDIC    '606162636465666768696A6B6C6D6E6F'X              */
 ASCII  = ASCII   || '2D2F626364656667686 96A2C256D3E3F'X
/*                                                                  */
/*       character                         : # @ ' = "              */
/*       EBCDIC    '707172737475767778797A7B7C7D7E7F'X              */
 ASCII  = ASCII   || '707172737475767778793A2340273D22'X
/*                                                                  */
/*       character     a b c d e f g h i                            */
/*       EBCDIC    '808182838485868788898A8B8C8D8E8F'X              */
 ASCII  = ASCII   || '806162636465666768698A8B8C8D8E8F'X
/*                                                                  */
/*       character     j k l m n o p q r                            */
/*       EBCDIC    '909192939495969798999A9B9C9D9E9F'X              */
 ASCII  = ASCII   || '906A6B6C6D6E6F7071729A9B9C9D9E9F'X
/*                                                                  */
/*       character       s t u v w x y z                            */
/*       EBCDIC    'A0A1A2A3A4A5A6A7A8A9AAABACADAEAF'X              */
 ASCII  = ASCII   || 'A0A17374757677787 97AAAABACADAEAF'X
/*                                                                  */
/*       character                                                  */
/*       EBCDIC    'B0B1B2B3B4B5B6B7B8B9BABBBCBDBEBF'X              */
 ASCII  = ASCII   || 'B0B1B2B3B4B5B6B7B8B9BABBBCBDBEBF'X
/*                                                                  */
/*       character     A B C D E F G H I                            */
/*       EBCDIC    'C0C1C2C3C4C5C6C7C8C9CACBCCCDCECF'X              */
 ASCII  = ASCII   || 'C041424344454647 84 9CACBCCCDCECF'X
/*                                                                  */
/*       character     J K L M N O P Q R                            */
/*       EBCDIC    'D0D1D2D3D4D5D6D7D8D9DADBDCDDDEDF'X              */
 ASCII  = ASCII   || 'D04A4B4C4D4E4F505152DADBDCDDDEDF'X
/*                                                                  */
/*       character       S T U V W X Y Z                            */
/*       EBCDIC    'E0E1E2E3E4E5E6E7E8E9EAEBECEDEEEF'X              */
 ASCII  = ASCII   || 'E0E1535455565758595AEAEBECEDEEEF'X
/*                                                                  */
/*       character   0 1 2 3 4 5 6 7 8 9                            */
/*       EBCDIC    'F0F1F2F3F4F5F6F7F8F9FAFBFCFDFEFF'X              */
 ASCII  = ASCII   || '30313233343536373839FAFBFCFDFEFF'X
 RETURN ASCII
```

ASCII to EBCDIC

```
/*                                                                   */
/* Procedure to create a translation string for conversion from      */
/* ASCII to EBCDIC                                                   */
/*                                                                   */
/* Usage: EBCDIC = A2E()                                             */
/*        string = TRANSLATE(ascii string,EBCDIC)                    */
/*                                                                   */
/*                                                                   */
/*        character                                                  */
/*        ASCII     '000102030405060708090A0B0C0D0E0F'X              */
 EBCDIC =           '000102030405060708090A0B0C0D0E0F'X
/*                                                                   */
/*        character                                                  */
/*        ASCII     '101112131415161718191A1B1C1D1E1F'X              */
 EBCDIC = EBCDIC || '101112131415161718191A1B1C1D1E1F'X
/*                                                                   */
/*        character   b ! " # $ % & ' ( ) * + , - . /                */
/*        ASCII     '20212223242526272829 2A2B2C2D2E2F'X             */
 EBCDIC = EBCDIC || '405A7F7B5B6C267D4D5D5C4E6B604B61'X
/*                                                                   */
/*        character   0 1 2 3 4 5 6 7 8 9 : ; < = > ?                */
/*        ASCII     '303132333435363738393A3B3C3D3E3F'X              */
 EBCDIC = EBCDIC || 'F0F1F2F3F4F5F6F7F8F97A5E4C7E6E6F'X
/*                                                                   */
/*        character   @ A B C D E F G H I J K L M N O                */
/*        ASCII     '4041424344454647 48494A4B4C4D4E4F'X             */
 EBCDIC = EBCDIC || '7CC1C2C3C4C5C6C7C8C9D1D2D3D4D5D6'X
/*                                                                   */
/*        character   P Q R S T U V W X Y Z                          */
/*        ASCII     '505152535455565758595A5B5C5D5E5F'X              */
 EBCDIC = EBCDIC || 'D7D8D9E2E3E4E5E6E7E8E95B5C5D5E5F'X
/*                                                                   */
/*        character     a b c d e f g h i j k l m n o                */
/*        ASCII     '6061626364656667 68696A6B6C6D6E6F'X             */
 EBCDIC = EBCDIC || '60818283848586878889919293949596'X
/*                                                                   */
/*        character   p q r s t u v w x y z                          */
/*        ASCII     '7071727374757677 78797A7B7C7D7E7F'X             */
 EBCDIC = EBCDIC || '979899A2A3A4A5A6A7A8A97B7C7D7E7F'X
/*                                                                   */
/*        character                                                  */
/*        ASCII     '8081828384858687 88898A8B8C8D8E8F'X             */
 EBCDIC = EBCDIC || '8081828384858687 88898A8B8C8D8E8F'X
/*                                                                   */
/*        character                                                  */
/*        ASCII     '9091929394959697 98999A9B9C9D9E9F'X             */
 EBCDIC = EBCDIC || '9091929394959697 98999A9B9C9D9E9F'X
/*                                                                   */
/*        character                                                  */
/*        ASCII     'A0A1A2A3A4A5A6A7 A8A9AAABACADAEAF'X             */
 EBCDIC = EBCDIC || 'A0A1A2A3A4A5A6A7 A8A9AAABACADAEAF'X
/*                                                                   */
/*        character                                                  */
/*        ASCII     'B0B1B2B3B4B5B6B7 B8B9BABBBCBDBEBF'X             */
 EBCDIC = EBCDIC || 'B0B1B2B3B4B5B6B7 B8B9BABBBCBDBEBF'X
/*                                                                   */
/*        character                                                  */
/*        ASCII     'C0C1C2C3C4C5C6C7 C8C9CACBCCCDCECF'X             */
```

```
 EBCDIC = EBCDIC || 'C0C1C2C3C4C5C6C7C8C9CACBCCCDCECF'X
/*                                                                  */
/*        character                                                 */
/*        ASCII     'D0D1D2D3D4D5D6D7D8D9DADBDCDDDEDF'X             */
 EBCDIC = EBCDIC || 'D0D1D2D3D4D5D6D7D8D9DADBDCDDDEDF'X
/*                                                                  */
/*        character                                                 */
/*        ASCII     'E0E1E2E3E4E5E6E7E8E9EAEBECEDEEEF'X             */
 EBCDIC = EBCDIC || 'E0E1E2E3E4E5E6E7E8E9EAEBECEDEEEF'X
/*                                                                  */
/*        character                                                 */
/*        ASCII     'F0F1F2F3F4F5F6F7F8F9FAFBFCFDFEFF'X             */
 EBCDIC = EBCDIC || 'F0F1F2F3F4F5F6F7F8F9FAFBFCFDFEFF'X
 RETURN EBCDIC
```

A.3.1.8 Stock Inquiry Program for TSO

This program uses the following subroutines:

- A.3.1.5, "Stock Inquiry Parameter Prompter" on page 97.
- A.3.1.9, "APPC Parameter Read for TSO."

```
/* REXX */
/* Stock Inquiry - TSO                                              */
/*                                                                  */
/* This transaction drives the IMS Stock Inquiry transaction, which */
/* returns data about stocks and stock indices.                     */
/*                                                                  */
/* Logic                                                            */
/*                                                                  */
/* Get request(s) from the user                                     */
/* Drive the IMS transaction                                        */
/* Show the replies ro the user                                     */
/*  - this involves a graphical display of the data                 */
/*                                                                  */
/* Parse user's request                                             */
/*                                                                  */
PARSE ARG REQUEST NAME
OUTMSG   = STOCKICM(REQUEST, NAME)
/*                                                                  */
/* Do the APPC process                                              */
/*                                                                  */
RESPONSE = APPCTSO('STOCKI', OUTMSG)
SAY RESPONSE
/*                                                                  */
/* This is where the fancy graphics would be done                   */
/*                                                                  */
EXIT
```

A.3.1.9 APPC Parameter Read for TSO

This program uses A.3.1.3, "APPC Driver" on page 94.

```
/* REXX */
/*                                                                  */
/* Subroutine to send an APPC message to TRANNAME and elicit a      */
/* response.                                                        */
/*                                                                  */
PARSE ARG TRANNAME, OUTMSG
/*                                                                  */
/* Allocate the file of APPC parameters.                            */
/*                                                                  */
```

```
ADDRESS TSO "ALLOC F(INPUT) DA('IMS510.STOCK.DATA(LOCAL)') SHR REUSE"
/*                                                                  */
/* Process until end of file, or non-comment found                  */
/*                                                                  */
'EXECIO 1 DISKR INPUT (STEM INPARM OPEN)'
DO WHILE RC=0
   IF SUBSTR(INPARM1,1,1) \= '*' THEN LEAVE
   'EXECIO 1 DISKR INPUT (STEM INPARM)'
END
'EXECIO 0 DISKR INPUT (FINIS)'
ADDRESS TSO "FREE F(INPUT)"
/*                                                                  */
/* Decode LU 6.2 parameters, dropping blanks where necessary        */
/*                                                                  */
LUNAME   = SUBSTR(INPARM1,01,17) /* IMS's LU name                  */
LOGMODE  = SUBSTR(INPARM1,18,08) /* VTAM logmode                   */
SIDEINFO = SUBSTR(INPARM1,26,08) /* APPC side info                 */
STOCKI   = SUBSTR(INPARM1,34,08) /* txn for Stock Inquiry          */
CDEAL    = SUBSTR(INPARM1,42,08) /* txn for Deal Notify            */
SELECT
  WHEN (TRANNAME) = 'STOCKI' THEN TPN = STOCKI
  WHEN (TRANNAME) = 'CDEAL'  THEN TPN = CDEAL
  OTHERWISE EXIT
END
/*                                                                  */
/* Call the APPC driver                                             */
/*                                                                  */
RETURN APPCDRV(LUNAME,LOGMODE,SIDEINFO,TPN,OUTMSG)
```

A.3.2 Deal Closed Functions

This function notifies the central Stock Exchange System of deals.

A.3.2.1 Deal Closed PSB

```
*
* Central Deal notification MPP
*
TPPCB1    PCB TYPE=TP,MODIFY=YES
CONTROL   PCB TYPE=DB,NAME=CNTL&SUFF,PROCOPT=GOT,KEYLEN=8
          SENSEG NAME=CONTROL,PARENT=0
STOCK     PCB TYPE=DB,NAME=STOCK&SUFF,PROCOPT=IR,KEYLEN=8
          SENSEG NAME=STOCK,PARENT=0
          SENSEG NAME=TXNLOG,PARENT=STOCK,PROCOPT=I
*
          PSBGEN PSBNAME=CDEAL&SUFF,LANG=ASSEM,CMPAT=YES
          END
```

A.3.2.2 Deal Closed Program for IMS

This program uses

- A.3.1.3, "APPC Driver" on page 94.
- A.3.2.3, "APPC Change Call" on page 105.

```
/* Central deal notification - IMS                                  */
/*                                                                  */
/* Although this is a new application, we have chosen to implement  */
/* it using the implicit LU 6.2 interface. This is principally to   */
/* speed the rescheduling of this transaction, which we expect to be */
/* performance critical.                                            */
/*                                                                  */
```

```
/* Central deal notification sends asynchronous messages to the      */
/* central stock exchange system. It is triggered asynchronously by  */
/* the end user when he confirms a deal has having taken place.      */
/*                                                                   */
/* Logic                                                             */
/*                                                                   */
/* Get message from queue                                            */
/* Get central system details (segment 'SE' from control DB)         */
/* While still messages                                              */
/*   Modify quantity of stock held (STOCK segment, FLD call)         */
/*   Log details of the deal on stock DB (TXNLOG segment)            */
/*   Send summary data to central system, via modifiable TP PCB      */
/*   Get another message from queue                                  */
/* End of while                                                      */
/*                                                                   */
address REXXIMS
'IMSRXTRC 0'                              /* production trace setting */
/*                                                                   */
/* Layout of stock exchange segment from the stock database          */
/*                                                                   */
MAP = '.  C 4 : TPN C 8 : SIDEINFO C 8 : LOGMODE C 8 : LUNAME C 8'
'MAPDEF SE     MAP REPLACE'
/*                                                                   */
/* Layout of transaction log segment from the control database       */
/*                                                                   */
MAP = 'LL    B 2 : STOCK_NAME C  8 : ACTION C 1 : QUANTITY P 6 : ',
      'PRICE P 4 : DEALER     C 40'
'MAPDEF TXNLOG MAP REPLACE'
/*                                                                   */
/* Layout of field search arguments                                  */
/*                                                                   */
/* 1.FSA to increase quantity                                        */
/*                                                                   */
MAP= 'NAME C 8 : STAT  C 1 : OPCODE  C 1 : QUANTITY P 6 : END   C 1'
'MAPDEF FSA1   MAP REPLACE'
NAME  = 'QUANTITY'
STAT  = ' '; OPCODE  = '+'; END   = ' '
/*                                                                   */
/* 2.FSA to decrease quantity and check it doesn't go negative       */
/*                                                                   */
MAP= 'NAME C 8 : STAT1 C 1 : OPCODE1 C 1 : QUANTITY P 6 : CONN  C 1 : ',
     'NAME C 8 : STAT2 C 1 : OPCODE2 C 1 : QUANTITY P 6 : END   C 1'
'MAPDEF FSA2   MAP REPLACE'
STAT1 = ' '; OPCODE1 = 'H'; CONN  = '*'
STAT2 = ' '; OPCODE2 = '-'
/*                                                                    */
/* Start of logic: Get Stock Exchange System details                  */
/*                                                                    */
SSA = 'CONTROL (TYPE    = SE)'
'GU CONTROL *SE SSA'
/*                                                                    */
/* Get Message                                                        */
/*                                                                    */
'GU IOPCB INMSG'
DO WHILE IMSQUERY('STATUS') = '  '
   /*                                                                 */
   /* Decypher input message                                          */
   /*                                                                 */
   PARSE VAR INMSG . STOCK_NAME ACTION QUANTITY PRICE HHMMSST DEALER
```

```
   /*                                                                  */
   /* Update stock held                                                */
   /* This is a field call to reduce the length of time that the       */
   /* STOCK segment is locked. The next peice of logic will lock the   */
   /* that segment, but the sync call is shortly after that.           */
   /*                                                                  */
   ACTION     = TRANSLATE(ACTION)           /* converts to upper case */
   STOCK_NAME = TRANSLATE(STOCK_NAME)       /* converts to upper case */
   SSA1 = 'STOCK   (NAME    = ' || SUBSTR(STOCK_NAME,1,8,' ') || ')'
   IF ACTION = 'B'
   THEN 'FLD  STOCK *FSA1 SSA1'
   ELSE 'FLD  STOCK *FSA2 SSA1'
   /*                                                                  */
   /* Complete the TXNLOG segment                                      */
   /*                                                                  */
   LL     = LENGTH(DEALER) + 21      /* calculate true segment length */
   /*                                                                  */
   /* Log the transaction details                                      */
   /*                                                                  */
   SSA2   = 'TXNLOG   '
   'ISRT STOCK *TXNLOG SSA1 SSA2'
   /*                                                                  */
   /* Send summary data to Stock exchange system                       */
   /*                                                                  */
   CALL APPCCHNG LUNAME LOGMODE TPN SIDEINFO
   OUTMSG = STOCK_NAME PRICE QUANTITY HHMMSST
   'ISRT TPPCB1 OUTMSG'
   /*                                                                  */
   /* Acknowledge to user                                              */
   /*                                                                  */
   IF ACTION = B THEN DO; ACT = 'bought'; PREP ='from'; END
                  ELSE DO; ACT = 'sold'  ; PREP ='to'  ; END
   RESPONSE = QUANTITY STOCK_NAME 'shares' ACT 'at' PRICE PREP DEALER
   'ISRT IOPCB RESPONSE'
   /*                                                                  */
   /* Get another message (implicit PURG call as well)                 */
   /*                                                                  */
   'GU IOPCB INMSG'
END
EXIT
```

A.3.2.3 APPC Change Call

This subroutine make APPC change calls for a number of functions. We list it here, because this is the first reference.

```
/*                                                                  */
/* Subroutine to issue an APPC change call.                         */
/* IMS is fussy about the format of the options, so this routine    */
/* handles all the reformatting                                     */
/*                                                                  */
PARSE ARG LUNAME LOGMODE TPNAME SIDEINFO .
address REXXIMS
/*                                                                  */
/* Initialise options string                                        */
/*                                                                  */
OPTIONS    = ''
/*                                                                  */
/* Initialise flag which says we are the first parameter            */
/* (and therefore should not have a leading comma)                  */
```

```
/*                                                                  */
FIRST      = 'Y'
/*                                                                  */
/* Process VTAM LU name, if specifed.                               */
/*                                                                  */
LUNAME     = STRIP(LUNAME,BOTH)    /* drop leading + trailing blanks */
IF LENGTH(LUNAME)      ¬= 0 THEN DO
   OPTIONS =  OPTIONS || 'LU='     || LUNAME
   FIRST   = 'N'              /* other parameters are not the first  */
END
/*                                                                  */
/* Process VTAM log mode, if specified.                             */
/* Note the logic to add a comma if this isn't the first parameter  */
/* and reset a flag if it is the first parameter                    */
/*                                                                  */
LOGMODE    = STRIP(LOGMODE,BOTH)   /* drop leading + trailing blanks */
IF LENGTH(LOGMODE)     ¬= 0 THEN DO
   IF FIRST = 'Y' THEN FIRST = 'N'; ELSE OPTIONS = OPTIONS || ','
   OPTIONS  =  OPTIONS || 'MODE=' || LOGMODE
END
/*                                                                  */
/* Process TP name, if specified.                                   */
/* Note the logic to add a comma if this isn't the first parameter  */
/* and reset a flag if it is the first parameter                    */
/* Also note that IMS requires an LL field in front of TP name.     */
/*                                                                  */
TPNAME     = STRIP(TPNAME,BOTH)    /* drop leading + trailing blanks */
IF LENGTH(TPNAME) ¬= 0 THEN DO
   IF FIRST = 'Y' THEN FIRST = 'N'; ELSE OPTIONS = OPTIONS || ','
   LEN = D2C(LENGTH(TPNAME)+2,2)
   OPTIONS =  OPTIONS || 'TPN=' ||  LEN || TPNAME
END
/*                                                                  */
/* Process MVS side information, if specified.                      */
/* Note the logic to add a comma if this isn't the first parameter  */
/* and reset a flag if it is the first parameter                    */
/*                                                                  */
SIDEINFO   = STRIP(SIDEINFO,BOTH)  /* drop leading + trailing blanks */
IF LENGTH(SIDEINFO)    ¬= 0 THEN DO
   IF FIRST = 'Y' THEN FIRST = 'N'; ELSE OPTIONS = OPTIONS || ','
   OPTIONS  =  OPTIONS || 'SIDE=' || SIDEINFO
END
/*                                                                  */
'CHNG TPPCB1 DFSLU62  OPTIONS ERRORMSG'
EXIT
```

A.3.2.4 Deal Closed Program for Workstation

This program uses the following subroutines:

- A.3.2.5, "Deal Closed Parameter Prompter" on page 107.
- A.3.1.6, "APPC Parameter Read for Workstation" on page 98.

```
/* Central deal notification - PWS                                  */
/*                                                                  */
/* This PC function sends a message to IMS to say that a deal has   */
/* been struck. IMS acknowledges the transaction using LU 6.2. No   */
/* data is returned. The IMS transaction uses the implicit LU 6.2   */
/* interface, blissfully unaware of APPC.                           */
/*                                                                  */
/*                                                                  */
```

```
/* Get and decode user's request                                    */
/*                                                                  */
PARSE ARG STOCK BUY QTY PRICE DEALER
OUTMSG = CDEALCOM(STOCK, BUY, QTY, PRICE, DEALER)
/*                                                                  */
/* Do the APPC processing and report to the user                    */
/*                                                                  */
RESPONSE = APPCPC('CDEAL', OUTMSG)
SAY RESPONSE
EXIT
```

A.3.2.5 Deal Closed Parameter Prompter

This subroutine prompts the user for parameters needed by Deal Closed.

```
/* REXX */
/* Central deal notification - common processing                    */
/*                                                                  */
/* This function decodes parameters for Central Deal Notification   */
/* and prompts the user for any that are missing. It then builds    */
/* them into a message for IMS.                                     */
/*                                                                  */
/* Interface Layout (message to IMS)                                */
/*    1.Transaction Code                                            */
/*    2.Name of stock                                               */
/*    3.B (= buy) or S (= sell)                                     */
/*    4.quantity of shares bought or sold                           */
/*    5.price at which the deal was done                            */
/*    6.time at which the deal was done                             */
/*    7.name of dealer we traded with                               */
/*                                                                  */
/* Logic                                                            */
/*                                                                  */
/* Get request(s) from the user                                     */
/* Format into a message for IMS                                    */
/*                                                                  */
/*                                                                  */
PARSE ARG STOCK, BUY, QTY, PRICE, DEALER
/*                                                                  */
IF STOCK = '' THEN DO
   SAY "Stock name?"
   PULL STOCK
END
STOCK = TRANSLATE(STOCK)                  /* converts to upper case */
/*                                                                  */
BUY = TRANSLATE(LEFT(BUY,1))
DO WHILE BUY \= 'S' & BUY \= 'B'
   SAY "Buy or Sell?"
   PULL BUY
   BUY = TRANSLATE(LEFT(BUY,1))           /* converts to upper case */
END
IF BUY = 'B' THEN DO; ACTION = 'bought'; PREP = 'from'; END
ELSE DO; ACTION = 'sold'; PREP = 'to'; END
/*                                                                  */
IF QTY   = '' THEN DO
   SAY "Quantity traded?"
   PULL QTY
END
/*                                                                  */
IF PRICE = '' THEN DO
```

```
   SAY "Price traded at?"
   PULL PRICE
END
/*                                                                  */
IF DEALER = '' THEN DO
   SAY ACTION PREP "whom?"
   PULL DEALER
END
RETURN  STOCK BUY QTY PRICE HHMMSST DEALER
```

A.3.2.6 Deal Closed Program for TSO

This program uses the following subroutines:

- A.3.2.5, "Deal Closed Parameter Prompter" on page 107.
- A.3.1.9, "APPC Parameter Read for TSO" on page 102.

```
/* REXX */
/* Central deal notification - TSO                                  */
/*                                                                  */
/* This TSO function sends a message to IMS to say that a deal has  */
/* been struck. IMS acknowledges the transaction using LU 6.2. No   */
/* data is returned. The IMS transaction uses the implicit LU 6.2   */
/* interface, blissfully unaware of APPC.                           */
/*                                                                  */
/*                                                                  */
/* Get user's request                                               */
/*                                                                  */
PARSE ARG STOCK BUY QTY PRICE DEALER
OUTMSG = CDEALCOM(STOCK, BUY, QTY, PRICE, DEALER)
/*                                                                  */
/* Do the APPC bits and reply to the user                           */
/*                                                                  */
RESPONSE = APPCTSO('CDEAL',  OUTMSG)
SAY RESPONSE
EXIT
```

A.3.3 Price Movement Functions

This function receives price information from the local Stock Exchange system.

A.3.3.1 Price Movements PSB

```
*
* Central price notification MPP
*
STOCK     PCB TYPE=DB,NAME=STOCK&SUFF,PROCOPT=IR,KEYLEN=16
          SENSEG NAME=STOCK,PARENT=0
          SENSEG NAME=INDEX,PARENT=STOCK
          SENSEG NAME=PRICE,PARENT=STOCK
INDEX     PCB TYPE=DB,NAME=INDEX&SUFF,PROCOPT=R,KEYLEN=8
          SENSEG NAME=INDEX,PARENT=0
*
          PSBGEN PSBNAME=CPRC&SUFF,LANG=ASSEM,CMPAT=YES
          END
```

A.3.3.2 Price Movements Program

```
/* Central price notify                                              */
/* We assume this is an old non-conversational MPP that is now       */
/* invoked by the central stock exchange system                      */
/*                                                                   */
/* Logic                                                             */
/*                                                                   */
/* Get message                                                       */
/* While there are still messages                                    */
/*   Read corresponding stock root                                   */
/*   (If the stock does not exist, a future enhancement could be to  */
/*    create it and notify the other offices.)                       */
/*   Calculate stock movement (+/-), we need it to update index(es)  */
/*   Update current price and insert a price movement                */
/*   Check this stock for belonging to stock index(es)               */
/*   Update stock index(es), do this with GHU/REPL                   */
/*     - update value and volume                                     */
/*     - check if this is a new high/low                             */
/*   Get message                                                     */
/* End of while                                                      */
/*                                                                   */
address REXXIMS
'IMSRXTRC 0'
/*                                                                   */
/* Layout of a stock segment                                         */
/*                                                                   */
MAP = 'SFIRST C 10 : PRICE P 4 : HIGH P 4 : LOW P 4 : SREST C 12'
'MAPDEF STOCK_SEG MAP REPLACE'
/*                                                                   */
/* Layout of a stock price segment                                   */
/*                                                                   */
MAP = 'HHMMSST P 4 : NEW_PRICE P 4 : QUANTITY P 6'
'MAPDEF PRICE_SEG MAP REPLACE'
/*                                                                   */
/* Layout of a stock index segment                                   */
/*                                                                   */
MAP = 'IFIRST C 10 : VAL P 4 : HIVAL P 4 : LOVAL P 4 : VOLUME P 6'
'MAPDEF INDEX_SEG MAP REPLACE'
/*                                                                   */
/* Get first message                                                 */
/*                                                                   */
'GU IOPCB INMSG'
DO WHILE IMSQUERY('STATUS') = '  '
   /*                                                               */
   /* Decipher input message                                        */
   /*                                                               */
   PARSE VAR INMSG . STOCK_NAME NEW_PRICE QUANTITY HHMMSST
   /*                                                               */
   /* Get the stock segment                                         */
   /*                                                               */
   SSA1 = 'STOCK   (NAME    = ' || SUBSTR(STOCK_NAME,1,8,' ') || ')'
   'GHU  STOCK *STOCK_SEG SSA1'
   PRICE_MOVEMENT = NEW_PRICE - PRICE
   /*                                                               */
   /* A possible refinement to this program would be dynamically to */
   /* create a Stock database record when we receive price          */
   /* information for an unknown stock.                             */
   /*                                                               */
   /* Check for new record highs and lows                           */
```

```
   /*                                                                */
   IF NEW_PRICE > HIGH THEN HIGH = NEW_PRICE
   IF NEW_PRICE < LOW  THEN LOW  = NEW_PRICE
   PRICE = NEW_PRICE
   SSA2  = 'STOCK    '
   'REPL STOCK *STOCK_SEG SSA2'
   /*                                                                */
   /* Insert a new price segment                                     */
   /*                                                                */
   SSA3 = 'PRICE    '
   'ISRT STOCK *PRICE_SEG SSA1 SSA3'
   /*                                                                */
   /* Process the stock index(es) this stock belongs to.             */
   /* All such index(es) will be local (because our data entry       */
   /* function ensure it).                                           */
   /*                                                                */
   SSA4 = 'INDEX    '
   'GU STOCK STOCK_INDEX SSA1 SSA4'
   DO WHILE IMSQUERY('STATUS') = '  '
      /*                                                             */
      /* Get Index from Index database                               */
      /*                                                             */
      SSA5 = 'INDEX   (NAME    = ' || STOCK_INDEX || ')'
      'GHU INDEX *INDEX_SEG SSA5'
      /*                                                             */
      /* Update index value and volume                               */
      /*                                                             */
      VAL    = VAL    + PRICE_MOVEMENT
      VOLUME = VOLUME + QUANTITY
      /*                                                             */
      /* Check for new record highs and lows                         */
      /*                                                             */
      IF VAL > HIVAL THEN HIVAL = VAL
      IF VAL < LOVAL THEN LOVAL = VAL
      /*                                                             */
      /* Replace the index                                           */
      /*                                                             */
      'REPL INDEX *INDEX_SEG SSA4'
      /*                                                             */
      /* Get next index (if any)                                     */
      /*                                                             */
      'GNP  STOCK STOCK_INDEX SSA4'
   END
   /*                                                                */
   /* Get another message (if any)                                   */
   /*                                                                */
   'GU IOPCB INMSG'
END
```

A.3.4 Market Close Functions

A.3.4.1 Market Close PSB

```
*
* Market Close BMP
*
TPPCB1    PCB TYPE=TP,MODIFY=YES             for close notification
STOCK     PCB TYPE=DB,NAME=STOCK&SUFF,PROCOPT=HA,KEYLEN=16
          SENSEG NAME=STOCK,PARENT=0
```

```
          SENSEG NAME=INDEX,PARENT=STOCK
          SENSEG NAME=MARKET,PARENT=STOCK
          SENSEG NAME=HISTORY,PARENT=STOCK
          SENSEG NAME=PRICE,PARENT=STOCK
INDEX     PCB TYPE=DB,NAME=INDEX&SUFF,PROCOPT=HA,KEYLEN=16
          SENSEG NAME=INDEX,PARENT=0
          SENSEG NAME=HISTORY,PARENT=INDEX
          SENSEG NAME=MARKET,PARENT=INDEX
CONTROL   PCB TYPE=DB,NAME=CNTL&SUFF,PROCOPT=GOT,KEYLEN=2
          SENSEG NAME=CONTROL,PARENT=0
MARKET    PCB TYPE=DB,NAME=MARKET&SUFF,PROCOPT=GOT,KEYLEN=8
          SENSEG NAME=MARKET,PARENT=0
*
          PSBGEN PSBNAME=CLOSE&SUFF,LANG=ASSEM,CMPAT=YES
          END
```

A.3.4.2 Market Close Program

This program uses A.3.2.3, “APPC Change Call” on page 105.

```
/* Market closing BMP                                                */
/*                                                                   */
/* Previous job steps perform the following tasks:                   */
/*   SDEP scan (into a sequential file for this program to read)     */
/*                                                                   */
/* Subsequent job steps perform the following tasks:                 */
/*   SDEP delete (subject to this program completing sucessfully)    */
/*                                                                   */
/* Logic                                                             */
/*                                                                   */
/* Note: This program is suitable for HSSP                           */
/*                                                                   */
/* For every market                                                  */
/*   send 'we have closed' message (change call to LU 6.2 device)    */
/*   (refer to Market Open/Close program for message layout.)        */
/*                                                                   */
/* Open file of extracted SDEP (transaction log) segments            */
/* For every transaction                                             */
/*   Accumulate quantity bought / sold by stock                      */
/*   Accumulate money spent / earned by stock                        */
/* End of tranaction analysis                                        */
/* Write a summary report                                            */
/*                                                                   */
/* Read Control DB to get history criteria                           */
/* Get todays date (used on all the history segments)                */
/* Calculate oldest retention dates (for later use)                  */
/*                                                                   */
/* Start at the beginning of the stock database                      */
/* For every local stock .....                                       */
/*   reconcile stock transactions with quantities on root            */
/*   update closing stock held if reconciled                         */
/*   If this stock is in an index, accumulate the value              */
/*   Initialize price history segment                                */
/*     Date (already known)                                          */
/*     Closing price to zero                                         */
/*     Daily high to zero                                            */
/*     Daily low to 9s                                               */
/*     Volume to zero                                                */
/*   Read first price movement,                                      */
/*   While still price movements ....                                */
```

```
/*      accumulate transaction volumes                                 */
/*      set running closing price                                      */
/*      check for new high or lows                                     */
/*      delete current price movement, get a new one.                  */
/*   End of while still price movements                                */
/*   Insert new history segment from summary of movements              */
/*   Delete old history segments up to their retention date            */
/*   Get next stock                                                    */
/*   If end of UoW ('GC' status) SYNC and re-read                      */
/* End of stock database process                                       */
/*                                                                     */
/* Read first index segment                                            */
/* For every local index .....                                        */
/*   Reconcile actual database value with freshly calculated value     */
/*   Update highest- and lowest-ever values if necessary               */
/*   Insert new history segment using reconciled value                 */
/*   Delete old history segments up to their retention date            */
/*   If end of UoW ('GC' status) SYNC and re-read                      */
/* End of index database process                                       */
/*                                                                     */
ADDRESS REXXIMS
'IMSRXTRC 0'                                    /* production value   */
/*                                                                     */
/* Local market segment (from Control DB)                              */
/*                                                                     */
MAP = '.  C 4 : LOCAL_MARKET C 8'
'MAPDEF LOCAL_SEG   MAP REPLACE'
/*                                                                     */
/* Retention criteria (from Control DB)                                */
/*                                                                     */
MAP = '.  C 4 : STOCK_DAYS P 2 : INDEX_DAYS P 2'
'MAPDEF RETN_SEG   MAP REPLACE'
/*                                                                     */
/* Market segment (from Market DB)                                     */
/*                                                                     */
MAP = 'MARKET_NAME  C 8 : .          C 3 : .       C 1 :',
      'SIDEINFO     C 8 : LOGMODE    C 8 : PARTNER C 8 :',
      '.            C 8 : STATUS_TXN C 8 : .       C 8'
'MAPDEF MARKET_SEG MAP REPLACE'
/*                                                                     */
/* Index segment (from Index DB)                                       */
/*                                                                     */
MAP = 'LL B 2 : INDEX_NAME C 8 : VAL P 4 : HIGH P 4 : LOW P 4 : VOL P 6'
'MAPDEF INDEX_SEG MAP REPLACE'
/*                                                                     */
/* Stock segment (from Stock DB)                                       */
/*                                                                     */
MAP = 'LL B 2 : STOCK_NAME C 8 : PRICE P 4 : . C 8: QTY P 6 : CLOSE P 6'
'MAPDEF STOCK_SEG MAP REPLACE'
/*                                                                     */
/* Transaction log segment (from Stock DB)                             */
/*                                                                     */
MAP = 'LL B 2 : STOCK_NAME C 8 : ACTION C 1 : QTY P 6 : PRICE P 4'
'MAPDEF TXNLOG_SEG MAP REPLACE'
/*                                                                     */
/* Price segment (from Stock DB)                                       */
/*                                                                     */
MAP = 'TIM P 4 : PRICE P 4 : QTY P 6'
'MAPDEF PRICE_SEG MAP REPLACE'
```

```
/*                                                                   */
/* History segment (from either Index or Stock DB)                   */
/*                                                                   */
MAP = 'DAT  P 5 : CLOSING P 4 : HIGH P 4 : LOW P 4 : VOL P 6'
'MAPDEF HISTORY_SEG MAP REPLACE'
/*                                                                   */
/* Start of Code, get local market name                              */
/*                                                                   */
SSA = 'CONTROL (TYPE    = LO)'
'GU CONTROL *LOCAL_SEG SSA'
OUT_TEXT = LOCAL_MARKET 'N'    /* text says: local market is closing */
/*                                                                   */
/* Send 'market closed' message to every market.                     */
/*                                                                   */
SSA = 'MARKET   '
'GU   MARKET *MARKET_SEG SSA'
OPEN = 'N'                                  /* notify market is closing*/
DO WHILE IMSQUERY('STATUS') = '  '
   /*                                                                */
   /* Use a LU6.2 change call for remote markets and                 */
   /* an ordinary change call for the local market.                  */
   /*                                                                */
   IF MARKET_NAME = LOCAL_MARKET
   THEN DO
      'CHNG  TPPCB1 STATUS_TXN'
      OUT_MESSAGE = STATUS_TXN OUT_TEXT
   END
   /*                                                                */
   /* With LU 6.2, IMS adds the transaction code for us.             */
   /*                                                                */
   ELSE DO
      CALL APPCCHNG PARTNER LOGMODE STATUS_TXN SIDEINFO
      OUT_MESSAGE = OUT_TEXT
   END
   /*                                                                */
   /* Send the 'we have opened' message on its way                   */
   /*                                                                */
   'ISRT TPPCB1 OUT_MESSAGE'
   'PURG TPPCB1'
   /*                                                                */
   /* Read the next MARKET segment                                   */
   /*                                                                */
   'GN MARKET *MARKET_SEG SSA'
END
STOCKS  = 0
INDEXES = 0
/*                                                                   */
/* Open file of extracted SDEP (transaction log) segments. If this   */
/* file were important, we would write to it using GSAM and          */
/* implement proper restart logic. As it is, EXECIO is sufficient.   */
/*                                                                   */
address MVS 'EXECIO 1 DISKR SDEPS (STEM SDEP OPEN)' /* open+read 1st */
/*                                                                   */
/* For every transaction                                             */
/*                                                                   */
DO WHILE RC = 0
   'MAPGET TXNLOG_SEG SDEP1'
   DO J = 1 TO STOCKS WHILE (STOCK_NAME \= STOCK_NAME.J)
   END
```

```
   IF J > STOCKS THEN DO
      STOCK_NAME.J = STOCK_NAME
      STOCK_QTY.J  = 0
      STOCK_VAL.J  = 0
      STOCKS = STOCKS + 1
   END
   /*                                                                */
   /* Accumulate quantity bought / sold                              */
   /* Accumulate money spent / earned                                */
   /*                                                                */
   IF ACTION = 'B'
   THEN DO
      STOCK_QTY.J = STOCK_QTY.J + QTY
      STOCK_VAL.J = STOCK_VAL.J + QTY * PRICE
   END
   ELSE DO
      STOCK_QTY.J = STOCK_QTY.J - QTY
      STOCK_VAL.J = STOCK_VAL.J - QTY * PRICE
   END
   address MVS 'EXECIO 1 DISKR SDEPS (STEM SDEP)'
END
/*                                                                */
/* End of transaction analysis                                    */
/*                                                                */
address MVS 'EXECIO 0 DISKR SDEPS (FINIS)' /* close file */
/*                                                                */
/* Write a summary report                                         */
/*                                                                */
ADDRESS MVS 'EXECIO 0 DISKW REPORT (OPEN)'
DO J = 1 TO STOCKS
   OUT1 = STOCK_NAME.J 'qty traded ' STOCK_QTY.J 'value' STOCK_VAL.J
   address MVS 'EXECIO 1 DISKW REPORT (STEM OUT)'
END
/*                                                                */
/* Read Control DB to get history criteria                        */
/*                                                                */
SSA = 'CONTROL (TYPE    = R1)'
'GU   CONTROL *RETN_SEG SSA'
/*                                                                */
/* Get todays date (used on all the history segments) and         */
/* Calculate oldest retention dates (for later use)               */
/*                                                                */
TODAY = DATE('S')                                 /* format YYYYMMDD*/
STOCK_RTN = DATE_CALC(TODAY,'-',STOCK_DAYS)
INDEX_RTN = DATE_CALC(TODAY,'-',INDEX_DAYS)
/*                                                                */
/* Start at the beginning of the stock database                   */
/*                                                                */
SSA1 = 'STOCK   *P '
SSA2 = 'MARKET  (NAME    = ' || LOCAL_MARKET || ')'
SSA3 = 'INDEX    '
SSA4 = 'PRICE    '
SSA5 = 'HISTORY  '
'GU   STOCK *STOCK_SEG  SSA1'
ST = IMSQUERY('STATUS')
DO WHILE ST = '  ' | ST = 'GC'
   /*                                                             */
   /* Take a sync point and re-issue our call, if we have reached */
   /* the end of a UoW                                            */
```

```
/*                                                                  */
IF ST = 'GC' THEN 'SYNC IOPCB'
ELSE DO
   /*                                                               */
   /* Check if stock is local (presence of a MARKET segment with  */
   /* local market name in it).                                    */
   /*                                                               */
   'GNP  STOCK STOCK_MARKET SSA2'
   IF IMSQUERY('STATUS') = '  ' THEN DO
      /*                                                            */
      /* Create a qualified SSA for this root.                     */
      /* Various processes below need it                           */
      /*                                                            */
      SSA6 = 'STOCK   (NAME    = ' || STOCK_NAME || ')'
      /*                                                            */
      /* Reconcile stock transactions with quantities on root      */
      /*                                                            */
      DO J = 1 TO STOCKS WHILE (STOCK_NAME.J \= STOCK_NAME)
      END
      /*                                                            */
      /* If this stock had transactions, reconcile them.           */
      /*                                                            */
      IF J <= STOCKS THEN DO
         /*                                                         */
         /* Update closing stock held if reconciled                */
         /*                                                         */
         'GHU  STOCK *STOCK_SEG  SSA6'
         IF QTY = CLOSE + STOCK_QTY.J
         THEN CLOSE = QTY
         ELSE DO
            OUT1 = STOCK_NAME 'reconciliation error',
                   QTY '(DB) \=' CLOSE + STOCK_QTY.J '(calc)'
            ADDRESS MVS 'EXECIO 1 DISKW REPORT (STEM OUT)'
         END
         'REPL STOCK *STOCK_SEG  SSA1'
      END
      /*                                                            */
      /* If this stock is in an index, accumulate the value        */
      /*                                                            */
      'GNP  STOCK STOCK_INDEX SSA3'
      DO WHILE IMSQUERY('STATUS') = '  '
         DO J = 1 TO INDEXES WHILE (INDEX_NAME.J \= STOCK_INDEX)
         END
         IF J > INDEXES THEN DO
            INDEX_NAME.J = STOCK_INDEX
            INDEX_VAL.J  = 0
            INDEXES = INDEXES + 1
         END
         INDEX_VAL.J = INDEX_VAL.J + PRICE
        'GNP  STOCK STOCK_INDEX SSA3'
      END
      /*                                                            */
      /* Initialize history segment                                */
      /*                                                            */
      DAT     = TODAY
      CLOSING = 0
      HIGH    = 0
      LOW     = 9999999
      VOL     = 0
```

```
            /*                                                          */
            /* Read first price movement                                */
            /*                                                          */
            'GHU STOCK *PRICE_SEG SSA6 SSA4'
            /*                                                          */
            /* While still price movements                              */
            /*                                                          */
            DO WHILE IMSQUERY('STATUS') = '  '
               VOL     = VOL + QTY        /* accumulate trade volume    */
               CLOSING = PRICE            /* set running close          */
               /*                                                       */
               /* Check for new highs and lows                          */
               /*                                                       */
               IF PRICE > HIGH THEN HIGH = PRICE
               IF PRICE < LOW  THEN LOW  = PRICE
               /*                                                       */
               /* Delete this and read next price movement              */
               /*                                                       */
               'DLET STOCK *PRICE_SEG'
               'GHU  STOCK *PRICE_SEG SSA6 SSA4'
            END
            /*                                                          */
            /* Insert new history segment                               */
            /*                                                          */
            'ISRT STOCK *HISTORY_SEG SSA6 SSA5'
            /*                                                          */
            /* Delete expired history segments                          */
            /*                                                          */
            'GHU  STOCK *HISTORY_SEG SSA6 SSA5'
            DO WHILE IMSQUERY('STATUS') = '  '
               /*                                                       */
               /* Check for expiration                                  */
               /*                                                       */
               IF DAT < STOCK_RTN THEN DO
                  /*                                                    */
                  /* Delete this and read next history                  */
                  /*                                                    */
                  'DLET STOCK *HISTORY_SEG'
                  'GHU  STOCK *HISTORY_SEG SSA6 SSA5'
               END
               ELSE LEAVE
            END
         END
      END
      /*                                                               */
      /* Get next stock                                                */
      /*                                                               */
      'GN   STOCK *STOCK_SEG  SSA1'
      ST = IMSQUERY('STATUS')
   END
   /*                                                                  */
   /* This is a good place for a SYNC call                             */
   /*                                                                  */
   'SYNC IOPCB'
   /*                                                                  */
   /* Start at the beginning of the index database                     */
   /*                                                                  */
   SSA1 = 'INDEX   *P '
   SSA2 = 'MARKET  (NAME    = ' || LOCAL_MARKET || ')'
```

```
SSA3 = 'HISTORY  '
'GU   INDEX *INDEX_SEG  SSA1'
ST = IMSQUERY('STATUS')
DO WHILE ST = '  ' | ST = 'GC'
   /*                                                              */
   /* Take a sync point and re-issue our call, if we have reached  */
   /* the end of a UOW                                             */
   /*                                                              */
   IF ST = 'GC' THEN 'SYNC IOPCB'
   ELSE DO
      /*                                                           */
      /* Check if index is local (presence of a MARKET segment with  */
      /* local market name in it).                                 */
      /*                                                           */
      'GNP  INDEX INDEX_MARKET SSA2'
      IF IMSQUERY('STATUS') = '  ' THEN DO
         /*                                                        */
         /* Create a qualified SSA for this root.                  */
         /* Various processes below need it                        */
         /*                                                        */
         SSA4 = 'INDEX   (NAME    = ' || INDEX_NAME || ')'
         /*                                                        */
         /* Reconcile index transactions with quantities on root   */
         /*                                                        */
         DO J = 1 TO INDEXES WHILE (INDEX_NAME.J \= INDEX_NAME)
         END
         IF J <= INDEXES THEN DO
            'GHU  INDEX *INDEX_SEG  SSA4'
            /*                                                     */
            /* Update closing index value if reconciled           */
            /*                                                     */
            IF VAL \= INDEX_VAL.J
            THEN DO
               OUT1 = INDEX_NAME 'reconciliation error',
                      VAL '(DB) \=' INDEX_VAL.J '(calc)'
               ADDRESS MVS 'EXECIO 1 DISKW REPORT (STEM OUT)'
            END
            VOL = 0                  /* reset volume for the new day */
            'REPL INDEX *INDEX_SEG  SSA1'
         END
         /*                                                        */
         /* Initialize history segment                             */
         /*                                                        */
         DAT     = TODAY
         CLOSING = VAL
         /*                                                        */
         /* Insert new history segment                             */
         /*                                                        */
         'ISRT INDEX *HISTORY_SEG SSA4 SSA3'
         /*                                                        */
         /* Delete expired history segments                        */
         /*                                                        */
         'GHU  INDEX *HISTORY_SEG SSA4 SSA3'
         DO WHILE IMSQUERY('STATUS') = '  '
            /*                                                     */
            /* Check for expiration                               */
            /*                                                     */
            IF DAT < INDEX_RTN THEN DO
               /*                                                  */
```

```
               /* Delete this and read next history                 */
               /*                                                   */
               'DLET INDEX *HISTORY_SEG'
               'GHU  INDEX *HISTORY_SEG SSA4 SSA3'
            END
            ELSE LEAVE
         END
      END
   END
   /*                                                               */
   /* Get next index                                                */
   /*                                                               */
   'GN   INDEX *INDEX_SEG  SSA1'
   ST = IMSQUERY('STATUS')
END
ADDRESS MVS 'EXECIO 0 DISKW REPORT (FINIS)'
/*                                                                  */
/* This is a good place for a SYNC call                             */
/*                                                                  */
'SYNC IOPCB'
EXIT
/*                                                                  */
/* Routine to calculate date differences                            */
/*                                                                  */
DATE_CALC: PROCEDURE
PARSE ARG DAT, OP, DAYS
MON.1 =31; MON.2 =28; MON.3 =31; MON.4 =30; MON.5 =31; MON.6 =30;
MON.7 =31; MON.8 =31; MON.9 =30; MON.10=31; MON.11=30; MON.12=31;
YEAR  = SUBSTR(DAT,1,4)
MONTH = SUBSTR(DAT,5,2)
DAY   = SUBSTR(DAT,7,2)
IF OP = '-' THEN DO
   DO WHILE DAYS > 0
      SELECT
         /*                                                        */
         /* If step backward is entirely within this month         */
         /*                                                        */
         WHEN (DAYS < DAY) THEN DO
            DAY  = DAY - DAYS            /* subtract number of days */
            DAYS = 0                     /* no days left to process */
         END
         /*                                                        */
         /* Step back a whole month                                */
         /*                                                        */
         WHEN (MONTH > 1) THEN DO
            IF YEAR // 4 = 0 THEN MON.2 = 29;ELSE MON.2 = 28;
            MONTH = MONTH - 1
            DAY   = DAY  + MON.MONTH     /* borrow a month of days  */
         END
         /*                                                        */
         /* Step back into December of the previous year           */
         /*                                                        */
         OTHERWISE DO
            MONTH = 12
            YEAR  = YEAR - 1
            DAY   = DAY  + 31            /* borrow December's days  */
         END
      END
```

```
   END
END
RETURN YEAR || RIGHT('00'||MONTH,2) || RIGHT('00'||DAY,2)
```

A.3.4.3 Market Close Job

This is sample JCL, showing two DEDB utilities and how to take an image copy using HSSP:

```
//*
//* MARKET CLOSE BMP - LONDON
//*
//*
//* DEDB SDEP SCAN UTILITY
//*
//SCAN     EXEC FPUTIL,DBD=STOCKL
//SYSIN     DD *
  TYPE SCAN
  AREA STOCK1L
  GO
  AREA STOCK2L
  GO
  AREA STOCK3L
  GO
//SCANCOPY  DD DISP=SHR,DSN=IMS510.STOCK.STOCKL.SDEPS
//*
//* MARKET CLOSE PROGRAM (USES HSSP)
//*
//CLOSE    EXEC IMSBATCH,MBR=CLOSE,PSB=CLOSEL
//SDEPS     DD DISP=SHR,DSN=IMS510.STOCK.STOCKL.SDEPS
//REPORT    DD SYSOUT=*
//DFSCTL    DD *                    HSSP OPTIONS
  SETO DB=STOCKL,PCB=STOCK,IC=2  /* REQUEST DUAL COPIES OF STOCK DB  */
  SETO DB=INDEXL,PCB=INDEX,IC=1  /* REQUEST ONE  COPY   OF INDEX DB  */
//*
//* DEDB SDEP DELETE UTILITY, IF EVERYTHING IS OK
//*
//         IF (RC = 0) THEN
//DELETE   EXEC FPUTIL,DBD=STOCKL
//SYSIN     DD *
  TYPE DELETE
  AREA STOCK1L
  GO
  AREA STOCK2L
  GO
  AREA STOCK3L
  GO
//         ENDIF
```

A.3.5 Market Open Functions

This function receives notice from the local Stock Exchange that it has opened.

A.3.5.1 Market Open PSB

```
*
* Market Open MPP
*
TPPCB1   PCB TYPE=TP,MODIFY=YES
CONTROL  PCB TYPE=DB,NAME=CNTL&SUFF,PROCOPT=GOT,KEYLEN=2
         SENSEG NAME=CONTROL,PARENT=0
MARKET   PCB TYPE=DB,NAME=MARKET&SUFF,PROCOPT=GOT,KEYLEN=8
         SENSEG NAME=MARKET,PARENT=0
*
         PSBGEN PSBNAME=OPEN&SUFF,LANG=ASSEM,CMPAT=YES
         END
```

A.3.5.2 Market Open Program

This program uses A.3.2.3, “APPC Change Call” on page 105.

```
/* Market opening MPP                                                 */
/*                                                                    */
/* This MPP is triggered by a message from the Local Stock Exchange   */
/* System. Conceptually, this is an LU 6.2 interface, although we     */
/* have chosen to use the implicit support.                           */
/*                                                                    */
/* Logic                                                              */
/*                                                                    */
/* Read local market name ('LO' segment from control DB)              */
/* Get input message (it has no interesting fields)                   */
/* While there are still messages ......                              */
/*    For every remote market ......                                  */
/*      send 'we have opened' message (change call to LU 6.2 device)  */
/*    For the local market, send via message switch                   */
/*    Get another input message                                       */
/*                                                                    */
Address REXXIMS
'IMSRXTRC 0'                                    /* production setting */
/*                                                                    */
/* Local market segment (from Control DB)                             */
/*                                                                    */
MAP = 'LL B 2 : . C 2 : LOCAL_MARKET C 8'
'MAPDEF CNTL_SEG   MAP REPLACE'
/*                                                                    */
/* Market segment (from Market DB)                                    */
/*                                                                    */
MAP = 'MARKET_NAME  C 8 : .          C 3 : .       C 1 :',
      'SIDEINFO     C 8 : LOGMODE    C 8 : PARTNER C 8 :',
      '.            C 8 : STATUS_TXN C 8 : .       C 8'
'MAPDEF MARKET_SEG MAP REPLACE'
/*                                                                    */
/* Start of Code, get local market name                               */
/*                                                                    */
SSA = 'CONTROL (TYPE    = LO)'
'GU CONTROL *CNTL_SEG SSA'
OUT_TEXT = LOCAL_MARKET 'Y'/* text of message, local market is opening*/
/*                                                                    */
/* Get input message (it has no interesting fields)                   */
/*                                                                    */
'GU IOPCB IN_MESSAGE'
DO WHILE IMSQUERY('STATUS') = '  '
   /*                                                                 */
   /* Get first market                                                */
   /*                                                                 */
```

```
   SSA = 'MARKET   '
   'GU MARKET *MARKET_SEG SSA'
   DO WHILE IMSQUERY('STATUS') = '  '
       /*                                                          */
       /* Use a LU6.2 change call for remote markets and           */
       /* an ordinary change call for the local market.            */
       /*                                                          */
       IF MARKET_NAME = LOCAL_MARKET
       THEN DO
          'CHNG' TPPCB1 STATUS_TXN
           OUT_MESSAGE = STATUS_TXN OUT_TEXT
       END
       /*                                                          */
       /* (Note: with an LU 6.2 call, IMS adds the trancode for us)*/
       /*                                                          */
       ELSE DO
          CALL APPCCHNG PARTNER LOGMODE STATUS_TXN SIDEINFO
          OUT_MESSAGE = OUT_TEXT
       END
       /*                                                          */
       /* Send the 'we have opened' message on its way             */
       /*                                                          */
       'ISRT TPPCB1 OUT_MESSAGE'
       'PURG TPPCB1'
       /*                                                          */
       /* Read the next MARKET segment                             */
       /*                                                          */
       'GN MARKET *MARKET_SEG SSA'
    END
    /*                                                          */
    /* Get the next message (there won't be one, but we need to set */
    /* a sync point)                                            */
    /*                                                          */
    'GU IOPCB IN_MESSAGE'
 END
 EXIT
```

A.3.6 Market Status Change Functions

This transaction changes the status of a market from open to closed or from closed to open. We do not describe it in Chapter 6, “Sample Application” on page 69 because it is part of the market open and close processes.

A.3.6.1 Market Status Change PSB

```
*
* Market Status change MPP
*
MARKET    PCB TYPE=DB,NAME=MARKET&SUFF,PROCOPT=R,KEYLEN=8
          SENSEG NAME=MARKET,PARENT=0
*
          PSBGEN PSBNAME=STATUS&SUFF,LANG=ASSEM,CMPAT=YES
          END
```

A.3.6.2 Market Status Change Program

```
/* Market status change (Open / Close) MPP                              */
/*                                                                      */
/* It updates a market's status depending on message content            */
/*                                                                      */
/* Logic                                                                */
/*                                                                      */
/* Get input message                                                    */
/* While there are messages to process                                  */
/*    Update market segment with new status                             */
/*    Get another message                                               */
/*                                                                      */
address REXXIMS
'IMSRXTRC 0'                                    /* production setting */
/*                                                                      */
/* FSA to set new status                                                */
/*                                                                      */
MAP = 'NAME C 8 : STATUS C 1 : OPCODE C 1 : VALUE C 1 : END C 1'
'MAPDEF FSA MAP REPLACE'
NAME = 'OPEN    ' ; STATUS = ' '; OPCODE = '='; END = ' ';
/*                                                                      */
/* Start of program, get a message                                      */
/*                                                                      */
'GU IOPCB MESSAGE'
/*                                                                      */
/* Process every message                                                */
/*                                                                      */
DO WHILE IMSQUERY('STATUS') = '  '
   PARSE VAR MESSAGE . REMOTE_MARKET OPEN
   SSA = 'MARKET  (NAME    = ' || SUBSTR(REMOTE_MARKET,1,8,' ') || ')'
   VALUE = SUBSTR(OPEN,1,1)
   'FLD MARKET *FSA SSA'
   /*                                                                   */
   /* Get the next message (there won't be one, but the call will      */
   /* cause a sync point)                                              */
   /*                                                                   */
   'GU IOPCB MESSAGE'
END
EXIT
```

A.3.7 Stock Exchange System

This program has no PSB source. We let IMS create a PSB for it by using the GPSB parameter of the APPLCTN macro in the stage one input. See A.4, “Stage One Macros” on page 125.

A.3.7.1 Stock Exchange Simulator Program

```
/* Stock exchange simulator                                             */
/* Conceptually, this is an APPC application, although we have chosen*/
/* to implement it using DLI calls.                                     */
/*                                                                      */
/* Deal Closed interface (messages from broker system)                  */
/*   1.Transaction code of this system                                  */
/*   2.Share name                                                       */
/*   3.Price at which deal was done                                     */
/*   4.Number of shares traded                                          */
/*   5.Time of trade                                                    */
/*                                                                      */
/* Price change information (message to broker system)                  */
```

```
/* Start Length Type Content                                          */
/*   1.Transaction code of broker system                              */
/*   2.Share name                                                     */
/*   3.New price                                                      */
/*   4.Number of shares traded                                        */
/*   5.Time of trade                                                  */
/*                                                                    */
/* Get message                                                        */
/* While still messages                                               */
/*   Insert message back to broker with the right transaction code    */
/*   Get another message                                              */
/* End of while                                                       */
/*                                                                    */
/* Define the layout of the input message                             */
/*                                                                    */
address REXXIMS
'IMSRXTRC 0'                                    /* production setting */
/*                                                                    */
'GU  IOPCB MESSAGE'            /* get a message to process            */
DO WHILE IMSQUERY('STATUS') = '  '      /* loop while still messages */
   PARSE VAR MESSAGE TRANCODE STOCK_NAME PRICE QUANTITY TIM
   /*                                                                 */
   /* Reformat trancode from 'SESYS' + suffix to 'CPRC' + suffix     */
   /* In a real application, the price transaction would be APPC and */
   /* the required parameters would be read from a database.         */
   /*                                                                 */
   TRANCODE = 'CPRC' || SUBSTR(TRANCODE,6,1) || '   '
   'CHNG TPPCB1 TRANCODE'
   MESSAGE = TRANCODE STOCK_NAME PRICE QUANTITY TIM
   'ISRT TPPCB1 MESSAGE'       /* send return message on its way     */
   /*                                                                 */
   /* Get next message to process                                     */
   /*                                                                 */
   'GU   IOPCB  MESSAGE'
END
EXIT
```

A.3.8 Testing Utilities

We wrote some programs to help correct odd situations that arose during testing.

A.3.8.1 Clean Up a Hanging Conversation

This program responds to an IMS transaction that is waiting for sync point to complete.

```
/* REXX */
PARSE ARG CONV TRACE
CONV = X2C(CONV)
ADDRESS CPICOMM
CALL ON ERROR NAME CPIC_FAIL
RESPONSE = ''
'CMRCV  CONV APPCMSG 1000 DR LEN SR RTS RC'
IF TRACE = 'TRACE' THEN
SAY 'Receive     ' DR SR LEN RC
DO I = 1 TO 10 WHILE RC = 0
   SELECT
      /*                                                            */
      /* If no data was returned, try for more                      */
```

```
      /*                                                              */
      WHEN (DR = 0) THEN DO                  /* no data returned     */
         'CMRCV  CONV APPCMSG 1000 DR LEN SR RTS RC'
         IF TRACE = 'TRACE' THEN
         SAY 'Receive     ' DR SR LEN RC
      END
      /*                                                              */
      /* If some data was returned, add it to the response and       */
      /* ask for more data.                                           */
      /*                                                              */
      WHEN (DR = 1 | DR = 2) THEN DO         /* data or partial data */
         IF LEN > 0 THEN DO
            RESPONSE = RESPONSE SUBSTR(APPCMSG,1,LEN)
            IF TRACE = 'TRACE' THEN
            SAY SUBSTR(APPCMSG,1,LEN)
         END
         'CMRCV  CONV APPCMSG 1000 DR LEN SR RTS RC'
         IF TRACE = 'TRACE' THEN
         SAY 'Receive     ' DR SR LEN RC
      END
      /*                                                              */
      /* If complete data was returned, add it to the response       */
      /*                                                              */
      WHEN (DR = 3) THEN                     /* complete data        */
         IF LEN > 0 THEN DO
            RESPONSE = RESPONSE SUBSTR(APPCMSG,1,LEN)
            IF TRACE = 'TRACE' THEN
            SAY SUBSTR(APPCMSG,1,LEN)
            LEAVE
         END
      OTHERWISE NOP                          /* no other values      */
   END
   /*                                                                 */
   /* If IMS asked for confirmation, supply it.                       */
   /*                                                                 */
   IF SR = 2 THEN DO
      'CMCFMD CONV RC'
       IF TRACE = 'TRACE' THEN
       SAY 'Confirmed   ' RC
       LEAVE
   END
END
RETURN RESPONSE
/*                                                                    */
/* Subroutine to trap CPIC errors, even if trace is off               */
/*                                                                    */
CPIC_FAIL: PROCEDURE EXPOSE RC
SAY CONDITION('D') RC
EXIT
```

A.3.8.2 Remove Unwanted IMS Messages

This program clears messages from the IMS message queue. It was particularly easy to use in this application because of the cloning techniques we used.

```
/* Program to clear messages from the message queue                  */
ADDRESS REXXIMS
'IMSRXTRC 0'
'GU IOPCB IN_MSG'
DO WHILE IMSQUERY('STATUS') = '  '
   'GU IOPCB IN_MSG'
END
EXIT
```

A.4 Stage One Macros

```
*
* Stage 1 macros for the Stock Market Application
*
* Databases, programs and transactions come in fours,
*   xxxxxx  - basic database name (not generated)
*   xxxxxxL - London version of the database
*   xxxxxxN - New York version of the database
*   xxxxxxT - Tokyo version of the database
*
          DATABASE ACCESS=UP,DBD=(STOCKL,STOCKN,STOCKT)
          DATABASE ACCESS=UP,DBD=(MARKETL,MARKETN,MARKETT)
          DATABASE ACCESS=UP,DBD=(INDEXL,INDEXN,INDEXT)
          DATABASE ACCESS=UP,DBD=(CURRL,CURRN,CURRT)
          DATABASE ACCESS=UP,DBD=(CNTLL,CNTLN,CNTLT)
*
* These databases are planned but not yet implemented
*
          DATABASE ACCESS=UP,DBD=(USERL,USERN,USERT)
          DATABASE ACCESS=UP,DBD=(NEWSL,NEWSN,NEWST)
*
* Some applications are Fast Path potential and so require two
* application macros, one with FPATH=YES and one without. These are
* distinguished by a suffix 'F' on the name of the Fast Path exclusive
* application
*
* This, combined with the need for 3 copies of everything, has led us
* to use instream macros to help with Stage 1 definitions.
*
* Transactions fall into several classes:
*   1.ordinary performance will do
*   2.performance not critical, but still important
*   3.performance critical
*
* Stock Inquiry - Synchronous LU 6.2 application driven from PWS or TSO
*
          MACRO
          STOCKI &SUFF=(L,N,T)
.*
          ACTR  255
&CTR      SETA  1
.STOCKI ANOP
.*
.* Fast Path Potential Transaction - MPP program
.*
          APPLCTN PSB=STOCKI&SUFF(&CTR),FPATH=NO,PGMTYPE=TP,           X
               SCHDTYP=PARALLEL
          TRANSACT CODE=STOCKI&SUFF(&CTR),EDIT=ULC,FPATH=YES,MODE=SNGL,X
```

```
               MAXRGN=3,PARLIM=20,                                    X
               MSGTYPE=(SNGLSEG,RESPONSE,2),PROCLIM=(50,5)
.*
.* Fast Path Potential Transaction - IFP program
.*
          APPLCTN PSB=STOCKI&SUFF(&CTR).F,FPATH=YES,PGMTYPE=TP,       X
               SCHDTYP=PARALLEL
          RTCODE CODE=STOCKI&SUFF(&CTR)
.* Loop  control logic
&CTR      SETA  &CTR+1
          AIF (&CTR LE N'&SUFF).STOCKI
          MEND
*
          STOCKI SUFF=(L,N,T)
          EJECT
*
* Central Price notification,
* Non-conversational MPP scheduled by local stock exchange system.
*
          MACRO
          CPRC   &SUFF=(L,N,T)
          ACTR  255
&CTR      SETA  1
.CPRC ANOP
          APPLCTN PSB=CPRC&SUFF(&CTR),FPATH=NO,PGMTYPE=TP,            X
               SCHDTYP=PARALLEL
          TRANSACT CODE=CPRC&SUFF(&CTR),EDIT=ULC,MODE=SNGL,           X
               MAXRGN=5,PARLIM=10,                                    X
               MSGTYPE=(SNGLSEG,NONRESPONSE,3),PROCLIM=(100,5)
.* Loop  control logic
&CTR      SETA  &CTR+1
          AIF (&CTR LE N'&SUFF).CPRC
          MEND
*
          CPRC SUFF=(L,N,T)
          EJECT
*
* Central Deal notification
* Non-conversational MPP scheduled by PWS or TSO
*
          MACRO
          CDEAL  &SUFF=(L,N,T)
          ACTR  255
&CTR      SETA  1
.CDEAL ANOP
          APPLCTN PSB=CDEAL&SUFF(&CTR),FPATH=NO,PGMTYPE=TP,           X
               SCHDTYP=PARALLEL
          TRANSACT CODE=CDEAL&SUFF(&CTR),EDIT=ULC,MODE=SNGL,          X
               MAXRGN=5,PARLIM=10,                                    X
               MSGTYPE=(SNGLSEG,NONRESPONSE,3),PROCLIM=(100,5)
.* Loop  control logic
&CTR      SETA  &CTR+1
          AIF (&CTR LE N'&SUFF).CDEAL
          MEND
*
          CDEAL SUFF=(L,N,T)
          EJECT
*
* Central Stock Exchange System
```

```
* Simulated by a non-conversational MPP
*
          MACRO
          SESYS  &SUFF=(L,N,T)
          ACTR  255
&CTR      SETA  1
.SESYS ANOP
          APPLCTN GPSB=SESYS&SUFF(&CTR),FPATH=NO,PGMTYPE=TP,           X
               SCHDTYP=PARALLEL
          TRANSACT CODE=SESYS&SUFF(&CTR),EDIT=ULC,MODE=SNGL,           X
               MAXRGN=5,PARLIM=10,                                     X
               MSGTYPE=(SNGLSEG,NONRESPONSE,3),PROCLIM=(100,5)
.* Loop  control logic
&CTR      SETA  &CTR+1
          AIF (&CTR LE N'&SUFF).SESYS
          MEND
*
          SESYS SUFF=(L,N,T)
          EJECT
*
* Market Status Update
* Non-conversational MPP scheduled by Market Open or Market Close
*
          MACRO
          STATUS &SUFF=(L,N,T)
          ACTR  255
&CTR      SETA  1
.STATUS ANOP
          APPLCTN PSB=STATUS&SUFF(&CTR),FPATH=NO,PGMTYPE=TP,           X
               SCHDTYP=PARALLEL
          TRANSACT CODE=STATUS&SUFF(&CTR),EDIT=ULC,MODE=SNGL,          X
               MAXRGN=5,PARLIM=10,                                     X
               MSGTYPE=(SNGLSEG,NONRESPONSE,1),PROCLIM=(100,5)
.* Loop  control logic
&CTR      SETA  &CTR+1
          AIF (&CTR LE N'&SUFF).STATUS
          MEND
*
          STATUS SUFF=(L,N,T)
          EJECT
*
* Market Open
* Non-conversational MPP scheduled by Local Stock Exchange
*
          MACRO
          OPEN &SUFF=(L,N,T)
          ACTR  255
&CTR      SETA  1
.OPEN ANOP
          APPLCTN PSB=OPEN&SUFF(&CTR),FPATH=NO,PGMTYPE=TP,             X
               SCHDTYP=PARALLEL
          TRANSACT CODE=OPEN&SUFF(&CTR),EDIT=ULC,MODE=SNGL,            X
               MAXRGN=5,PARLIM=10,                                     X
               MSGTYPE=(SNGLSEG,NONRESPONSE,1),PROCLIM=(100,5)
.* Loop  control logic
&CTR      SETA  &CTR+1
          AIF (&CTR LE N'&SUFF).OPEN
          MEND
*
```

```
          OPEN SUFF=(L,N,T)
          EJECT
*
* Market Close BMP
*
          APPLCTN PSB=CLOSEL,PGMTYPE=BATCH
          APPLCTN PSB=CLOSEN,PGMTYPE=BATCH
          APPLCTN PSB=CLOSET,PGMTYPE=BATCH
*
* Database Load BMPs
*
          APPLCTN PSB=DBLOADL,PGMTYPE=BATCH
          APPLCTN PSB=DBLOADN,PGMTYPE=BATCH
          APPLCTN PSB=DBLOADT,PGMTYPE=BATCH
*
* Utility BMPs, for issuing commands via DFSDDLTO.
* These also appear in the Security Gen input.
*
          APPLCTN GPSB=COMMANDL,PGMTYPE=BATCH
          TRANSACT CODE=COMMANDL
          APPLCTN GPSB=COMMANDN,PGMTYPE=BATCH
          TRANSACT CODE=COMMANDN
          APPLCTN GPSB=COMMANDT,PGMTYPE=BATCH
          TRANSACT CODE=COMMANDT
```

Appendix B. Using APPC with IMS

Throughout this chapter, we will be referring to examples in our sample application. If you would like more details about our this application, its description is in Chapter 6, “Sample Application” on page 69 and the coding details are in Appendix A, “Sample Application Code” on page 81.

B.1 APPC Concepts

APPC and LU6.2 are interchangeable terms. They are the name of a protocol which programs can use for communicating.

This communication is called a “conversation” because it is like a conversation between people. There are differences between an APPC conversation and a human conversation.

APPC conversations are between two programs. A program may take part in more than one conversation, but each conversation involves only two programs. These programs are called *partners* in the conversation.

APPC conversations are polite. A partner may not speak if it is listening and cannot listen when it is speaking. This does not stop a program from speaking and listening at the same time, provided it does so with different partners.

APPC uses the terms *sending* and *receiving* rather than speaking and listening.

APPC programs normally use VTAM for the communications.[18] Some of the APPC terms you will encounter are really VTAM terms.

B.1.1 APPC Terminology

There are many APPC terms you need to be familiar with.

B.1.1.1 Side Information

Side information is a set of information about a particular APPC program. It contains

Symbolic destination name	This is the key of the side information.
Logical Unit name	(or “LU name” for short) is a VTAM term for a node in the network. Typically, an APPC LU is a transaction manager such as IMS, CICS or APPC/MVS.
Mode name	This the name of a VTAM control block that describes the properties of the conversation.
Transaction Program name	(or *TP name* for short) is the name of the APPC program. In the case of IMS, the TP name is not a program, but a transaction. TP names can be 64 characters long.

[18] If you use APPC/MVS to converse with a partner on the same MVS system, APPC/MVS is clever enough not to use VTAM, but manage the conversation itself.

B.1.1.2 APPC/MVS TP Profile

APPC/MVS keeps transaction program attributes in its TP Profile data set. The profile is keyed on TP name. In the case of IMS, this information includes

- Transaction code (if different from the TP name)
- Transaction scheduling parameters (class and maximum regions)
- Security checking level.

B.2 Designing an APPC Application

There are a number of things to consider when you are designing an APPC application.

B.2.1 What Function on Which Platform?

A fundamental decision you will confront is which processing should be done on which platform. The decision is often based on the relative inconvenience of data redundancy versus poor response time.

B.2.1.1 An Example from Our Sample Application

The Stock Inquiry function in our sample application has a component that runs on a workstation and a component which runs on IMS. This function needs to know which stock market to go to for information about a particular stock. We considered three ways to do this:

- We could have chosen to let the workstation component decide which market to access. We considered two ways of doing this:
  - Trying every market to see if the stock trades there (this would be too slow).
  - Holding a cross reference of which stock trades where on every workstation (this would be a nightmare to maintain).
- We could have forced the user to choose which market to access. This has the advantages of speed and eliminating data redundancy.

  We decided against this option because we do not expect every user to know where every stock trades. There are many thousands of stocks and some trade on more than one market.
- In fact, we chose to let the IMS component decide (by consulting the stock database). Because of this, every stock in the world must have an entry on every copy of our stock database, a considerable amount of data redundancy.

  Fortunately, the fact that all the stock databases are similar makes the job of administering them easier. When a new stock appears or an old one disappears, you would make the same updates in every copy of the stock database. This means that the task could be automated.

B.2.2 Synchronization Level

You also have a choice of synchronization level (or sync level for short). There are three synchronization levels.

None — No synchronization support at all.

In this case, you cannot tell whether your partner has received a message or not.

Confirm — Confirmation of receipt.

Sync point — Full synchronization point support.

None of the platforms we are discussing support sync point at this time.

We recommend that you use sync level confirm. If you do this, IMS asks you to confirm that you have received its messages. This has implications when you are coding.

B.2.3 Which APPC Interface?

There are two interfaces you can use for APPC, depending on the platform your program runs on.

Note: You may also see references in the APPC literature to LU6.2 verbs (all starting MC_). These verbs are part of a metalanguage, but you cannot code using them.

B.2.3.1 CPIC

A variety of platforms support this interface. All our sample code uses CPIC calls, because this makes it portable between environments. We tested using TSO and IMS, but the code should work in other environments (for example OS/2).

CPIC calls can be tedious to program, because you have to set each conversation attribute with a separate CPIC call. You can see this in our sample code.

B.2.3.2 APPC/MVS

You can generally write an APPC program with fewer APPC/MVS calls than you need with CPIC. This is because you can specify conversation attributes on the APPC/MVS function calls. APPC/MVS also supplies a number of useful calls that are not part of CPIC.

However, you cannot take APPC/MVS calls to another platform. This might not be a problem if all the platforms you are interested in run under MVS. On the other hand, if you want to use workstations, APPC/MVS calls are not for you.

It would be a simple task to convert our sample application to APPC/MVS calls.

B.2.4 Implicit or Explicit APPC in IMS?

Another design choice is between using IMS or APPC for communication.

If you use IMS for communication, it handles all the APPC calls and your IMS program uses the IOPCB in the usual way. This is called the *implicit APPC interface*. Most of our examples use the implicit interface because they can receive messages from non APPC users (for example other programs).

On the other hand, you can use APPC calls within an IMS transaction. In this case IMS is not involved and you have to code all the APPC calls yourself. This is called the *explicit APPC interface*.

B.2.5 Synchronous or Asynchronous

You should spend some time deciding which functions to make synchronous and which should be asynchronous. This is particularly important when one of the partners is an IMS transaction.

B.2.5.1 With an IMS Transaction

You make the decision between synchronous and asynchronous when you code the CPIC calls that drive the IMS transaction. If you wait for a response (using the wait option of the APPC receive verb) the conversation is synchronous. If you deallocate the conversation before receiving a response the conversation is asynchronous. In this case, IMS will send the response later.

This is different from the way IMS forces synchronization with MODE=RESPONSE on the transaction definition.

B.2.5.2 Truly Synchronous

Using APPC, it is possible to have a truly synchronous conversation between IMS transactions. One of the transactions drives the other using explicit APPC calls. The driven transaction does not need to use explicit APPC calls, it can respond to the IOPCB and let IMS make the required APPC calls.

There is an example of how to code this in A.3.1.2, “Stock Inquiry Program for IMS” on page 88.

B.3 How to Drive an IMS Transaction

This section contains sample flows for driving an IMS transaction from a workstation program using different APPC techniques. We discuss the various combinations of the IMS APPC interface (implicit or explicit) versus conversation type (synchronous or asynchronous). We only discuss sync level confirm.

We indicate the differences between the various techniques to help you understand the advantages of using one over another.

Note: This is a difficult topic, please read it carefully.

B.3.1 Synchronous and Implicit

A synchronous APPC conversation between a workstation program and an IMS transaction using the implicit interface flows like this:

1. The workstation program
   - Allocates a conversation
   - Sends a message
   - Waits for a response.

   This process is the same for both synchronous flows.
2. IMS
   - Accepts the conversation
   - Receives the message
   - Confirms receipt
   - Schedules the transaction.

   The transaction need not know that the message came from APPC. It processes in the usual way:

- Gets the message using the IOPCB
- Processes
- Inserts a response to the IOPCB.

This process is common to both implicit flows.

3. The transaction has not finished. It is in the commit phase, waiting for confirmation. ***Thus, it is still holding locks.***

   IMS

   - Sends the response to the workstation.
   - Waits for confirmation.

   The workstation program

   - Receives the response.
   - Confirms to IMS.
   - Carries on with the rest of its processing.
   - Cannot continue the conversation.[19]

   IMS

   - Receives confirmation.
   - Completes the commit process.

   Only now does IMS release locks on the resources the transaction was using.

   This process is unique to the synchronous implicit flow.

You will find an example of this using CPIC calls in A.3.1.3, “APPC Driver” on page 94.

B.3.2 Synchronous and Explicit

A synchronous APPC conversation between a workstation program and an IMS transaction using the explicit interface flows like this:

1. The workstation program

   - Allocates a conversation
   - Sends a message
   - Waits for a response.

   This process is the same for both synchronous flows.

2. IMS

   - Recognizes that this is an explicit APPC application, because the transaction code is not in the IMSGEN.
   - Uses the APPC/MVS TP profile to decide the transaction attributes
   - Starts the transaction.

   The transaction

   - Accepts the conversation
   - Receives the message
   - Confirms receipt
   - Can use IMS facilities, by telling IMS which PSB it wants to use (using the **APSB** DLI call).

[19] Because the IMS transaction has already reached its commit point. If the workstation program needs to communicate it must start another conversation.

This process is common to both explicit flows.

3. The transaction

   - Processes
   - Sends a response
   - Waits for confirmation.

   The workstation program

   - Receives the response
   - Confirms the transaction
   - Carries on with the rest of its processing
   - Can continue the conversation, because the IMS transaction has not necessarily ended (it depends on how you code it).

   Either party may end the conversation by deallocating it.

   This process is unique to the synchronous explicit flow.

B.3.3 Asynchronous and Implicit

An asynchronous APPC conversation between a workstation program and an IMS transaction using the implicit interface flows like this:

1. The workstation program

   - Allocates a conversation
   - Sends a message
   - Deallocates the conversation
   - Waits for confirmation
   - Carries on processing (if necessary).

   This process is the same for both asynchronous flows.

2. IMS

   - Accepts the conversation
   - Receives the message
   - Confirms receipt
   - Schedules the transaction.

   The transaction need not know that the message came from APPC. It processes in the usual way:

   - Gets the message using the IOPCB
   - Processes
   - Inserts a response to the IOPCB.

   This process is common to both implicit flows.

3. There is no longer a conversation between the transaction and the workstation program because IMS has confirmed the deallocation request.

   A special IMS task

   - Allocates a fresh conversation (with DFSASYNC) or DFSCMD if you sent an IMS command rather than a transaction. Both DFSASYNC and DFSCMD are TP names. Your workstation should have programs with these names to process asynchronous IMS responses.
   - Sends the response
   - Waits for confirmation.

   DFSASYNC

- Runs on the workstation
- Receives the response
- Confirms to IMS
- Processes further (if necessary).

This process is unique to the asynchronous implicit flow.

B.3.4 Asynchronous and Explicit

An asynchronous APPC conversation between a workstation program and an IMS transaction using the explicit interface flows like this:

1. The workstation program

   - Allocates a conversation
   - Sends a message
   - Deallocates the conversation
   - Waits for confirmation
   - Carries on processing (if necessary).

   This process is the same for both asynchronous flows.

2. IMS

   - Recognizes that this is an explicit APPC application, because the transaction code is not in the IMSGEN.
   - Uses the APPC/MVS TP profile to decide the transaction attributes
   - Starts the transaction.

   The transaction

   - Accepts the conversation
   - Receives the message
   - Confirms receipt
   - Can use IMS facilities, by telling IMS which PSB it wants to use (using the **APSB** DLI call).

   This process is common to both explicit flows.

3. There is no longer a conversation between the transaction and the workstation program because the transaction has confirmed the deallocation request.

   The transaction

   - Processes
   - Allocates a fresh conversation that may involve a different workstation program.
   - Sends a response
   - Waits for confirmation.

   The workstation program

   - Receives the response
   - Confirms it
   - Processes further (if necessary).

   The transaction

   - Receives the confirmation
   - Carries on with the rest of its processing (if any).

   This process is unique to the asynchronous explicit flow.

B.3.5 APPC Coding and IMS

When you are processing responses from a transaction using the implicit APPC interface, you should continue issuing receive calls until IMS asks for confirmation. You can tell this because the “data received” indicator is set to 2. At this point you should supply confirmation (using the confirmed call).

You can send a message from an IMS transaction to any APPC program. You specify the APPC options on the change call. The APPC program could be another IMS transaction,perhaps on another IMS system. We have an example of such a change call in A.3.2.3, “APPC Change Call” on page 105.

There is an IMS service (DFSAPPC) that provides the same facility for a terminal user.

Please refer to *IMS/ESA V5 Application Programming: Transaction Manager*, SC26-8017 for the details of the change call and DFSAPPC. Specify the APPC options on the change call. The APPC program could be another IMS transaction,perhaps on another IMS system. We have an example of such a change call in A.3.2.3, “APPC Change Call” on page 105.

Please refer to *IMS/ESA V5 Application Programming: Transaction Manager*, SC26-8017 for the details of the change call.

B.4 Testing an IMS APPC Program

You test your IMS APPC programs in the same way as you would other IMS programs. The usual IMS testing tools do not recognize APPC calls and so cannot help with debugging these.

“Communication Manager/2” (also known as “CM/2”) provides a trace of CPIC calls on the workstation.

Other than that, you need to supply your own diagnostics.

B.4.1 Problems You Might Encounter

This is a list of problems we encountered while testing our sample application. It is not an exhaustive list of all the problems you might encounter.

B.4.1.1 ASCII and EBCDIC

When using a workstation, you need to convert between ASCII and EBCDIC. CM/2 automatically translates the LU name, TP name, and mode name, but you have to translate the messages yourself.

You can see how we addressed this problem in A.3.1.7, “Conversion between ASCII and EBCDIC” on page 99.

B.4.1.2 IMS Messages Not Arriving (DFS1964E)

If you send IMS an APPC message, but it does not start a transaction, you can check whether IMS has received it or not. Use the command

```
/DISPLAY LUNAME ims logical unit name TPNAME ALL
```

to show the transactions and their queues.

You might find that the transaction is queuing but not scheduling. You should find a DFS1964E message on the master and secondary consoles explaining why. One possible reason is that you have tried to schedule a response mode transaction using an asynchronous request (for example an APPC change call or DFSAPPC).

EMH transactions must be defined as response mode, so be sure to schedule them synchronously.

B.4.1.3 IMS and TP Name

When you start an IMS transaction using APPC (whether you use explicit or implicit APPC calls), IMS adds the TP name plus a blank to the front of the message. This can be a surprise if you are not expecting it.

In our sample application, we have several transactions that can be started using APPC or by a user at a terminal or by a program switch. We use REXX to parse a message into its constituent parts, rather than using the traditional method of column dependence.

B.4.1.4 Hanging Conversations (Waiting Sync Point)

During your testing, you may get into the situation where IMS is waiting for its APPC partner. You can see this from a /DISPLAY ACTIVE command because your region will have the status “WAITING SYNCPOINT.”

One way to get this is if you are using a synchronous conversation and have forgotten to confirm to IMS that you have received its response. The IMS transaction will wait indefinitely for a response, and will prevent IMS from shutting down. In this situation you can:

- Resume the conversation and supply the confirmation. To do this you have to know the conversation ID.

  This is the tidiest way to correct the problem. We used A.3.8.1, “Clean Up a Hanging Conversation” on page 123 for this function.

- If you were using TSO, log off and log on again (if you can).

  This ends the hanging conversation.

- Issue a /STOP REGION ... CANCEL command.

  This is the last resort, because the IMS control region may abend as a result of this command.

Appendix C. Using REXX with IMS

C.1 REXX and IMS

This section is a general discussion of REXX and IMS. We do not limit it to APPC topics.

You can find more about REXX and IMS in the part “IMS Adapter for REXX” in both *IMS/ESA V5 Application Programming: Database Manager*, SC26-8015 and *IMS/ESA V5 Application Programming: Transaction Manager*, SC26-8017.

C.1.1 Advantages of Using REXX with IMS

REXX has a number of advantages for the application developer:

- REXX programs are interpreted, so there is less time lag between making a code change and testing it. Interpretation also allows you to test incomplete programs in a way that is impossible with compiler-only languages. This increases programmer productivity, in spite of the fact that an interpreted program runs slower than a compiled one.

  Once you have tested your program, you can use the REXX compiler to create a production version.

- There is REXX support on a wide variety of platforms. This makes it easy to move the non-IMS components of your application from platform to platform.

- The IMS REXX adapter has a useful trace facility which is not available to other languages (see C.1.4, “IMS REXX Trace” on page 143).

C.1.2 Disadvantages of Using REXX with IMS

There are also some disadvantages to using REXX:

- Executing a REXX program is slow if the program is interpreted.

- Creating and initializing a REXX program is slow, even if you have compiled it. However, you can alleviate this problem; see C.1.3.5, “Region Considerations” on page 143 for more details.

  If you do not have the REXX compiler, you can define your REXX source library to library lookaside (LLA) which will improve the initialization performance.

  The initialization overhead is especially irksome in an MPP region, because many programs execute one after another and all have to initialize. In a DLI, BMP, or IFP region there is a single program and only one initialization overhead.

- The IMS REXX adapter restricts packed decimal fields to 11 digits (6 bytes).[20] This is awkward for applications that need high precision. You can code around this restriction, but it is tedious in the extreme.

[20] REXX itself supports only character data. Without the IMS adapter, you would find it difficult to support packed decimal at all.

C.1.3 Getting Started

This is a summary of how to prepare, execute and test a REXX program in IMS.

C.1.3.1 REXX Environments

IMS supports two REXX command environments: REXXIMS and REXXTDLI. REXXTDLI supports DL/I calls only, but REXXIMS supports DL/I calls and useful additional facilities. We have more to say about these in C.1.3.3, "Additional Facilities Provided for REXX" on page 141. All of our sample code uses REXXIMS, because we use some of the additional facilities.

You can use other environments as well:

MVS — MVS specific functions — Some of our sample programs use this for file I/O. This is because file I/O varies from platform to platform. In the Market Close BMP, we could have chosen to do file I/O with GSAM, but the contents of the file (run time messages) did not justify it.

CPICOMM — CPIC calls — All our sample programs use this for APPC communication.

LU62 — APPC/MVS versions of CPIC calls — None of our programs use these calls.

APPCMVS — Advanced APPC/MVS specific calls (not in CPIC) — None of our programs use these calls.

C.1.3.2 Coding a PSB and Using PCBs

You code a PSB for a REXX application in the same way as for any other language. You should specify LANG=ASSEM or LANG=COBOL.

If you use PCB labels or PCBNAME= parameters, you can reference the PCBs by name in your REXX source. If you do this, IMS uses the AIB interface for your calls. This can be important because IMS gives you more information (for example, the length of a fixed length segment) when you use the AIB interface. We use this throughout our sample application.

If you don't use PCB labels or PCBNAME= parameters,you have to use reference the PCBs by number using a "#" symbol. Except for the IOPCB, which always has a name of "IOPCB." If you use the PCB number, IMS uses the PCB interface.

For example, suppose we are using the sample application and we want to read the stock database. Assume that the PSB looks like this:

```
STOCK    PCB TYPE=DB,NAME=STOCK, ...
         SENSEG NAME=STOCK,PARENT=0

  ⋮

         PSBGEN LANG=ASSEM, ...
         END
```

Then we could retrieve a root segment using the AIB interface:

```
SSA = 'STOCK   (NAME    = IBM     )'
ADDRESS REXXTDLI 'GU STOCK segment SSA'
```

or using the PCB interface:

```
SSA = 'STOCK   (NAME    = IBM     )'
ADDRESS REXXTDLI 'GU #2 segment SSA'
```

C.1.3.3 Additional Facilities Provided for REXX

IMS provides a number of useful facilities of use with REXX. If you want further details, refer to the description of the individual keyword in the part "IMS Adapter for REXX" in both the *IMS/ESA V5 Application Programming: Database Manager*, SC26-8015. and *IMS/ESA V5 Application Programming: Transaction Manager*, SC26-8017.

The additional facilities fall into these five categories:

Mapping Facilities There are three REXXIMS commands for manipulating (or mapping) structured data. REXXIMS recognizes more data types than REXX itself and handles the conversion for you. These facilities are especially useful when you process database segments, which are usually highly structured and often use a variety of data types.

MAPDEF Defines a map. A map has a number of REXX variables defined over it. You can use a map instead of an I/O area, an SSA or an FSA.

MAPGET IMS assigns the REXX variables from the map. If you specify a map name as an output parameter on a DLI call, IMS will update the REXX variables from the map automatically.

MAPPUT IMS updates the map from the REXX variables. If you specify a map name as an input parameter on a DLI call, IMS will update the map from the REXX variables automatically.

The sample application uses the mapping functions extensively.

Information Facilities: There are three of these:

DLIINFO Provides information from the PCB (and the AIB if you are using the AIB interface). IMS returns this information in a string you will have to parse.

IMSQUERY A function which gives useful information about

- The last DLI call
- The current environment
- Storage obtained with the STORAGE command.

IMS returns only the information you ask for.

IMSRXTRC Sets the trace level (see C.1.4, "IMS REXX Trace" on page 143).

Our sample application uses the IMSQUERY('STATUS') function to check PCB status codes. This is more convenient than the DLIINFO command when you want only one piece of information.

Facilities for Explicit APPC Programs: There are two REXXIMS commands and two DLI calls for use by explicit APPC programs.

APSB Tells IMS what PSB to use. You use this call to allocate a PSB if you want to access any IMS resources.

DPSB Tells IMS you have finished with a PSB. You use this call after the commit point to deallocate a PSB you have previously allocated. You must use this call if you want to use more than one PSB.

SRRCMIT Tells IMS to commit your updates. It is the same as a SYNC call.

SRRBACK Tells IMS to back out your updates. It is the same as a ROLB call.

Our sample application does not use these facilities.

WTO Commands: There are three REXXIMS commands for sending messages and one for eliciting a response from the operator:

WTO Writes a message to the operator.
WTP Writes a message to the program.
WTL Writes a message to the MVS console log.
WTOR Writes a message to the operator and returns the response.

Our sample application does not use these commands.

Miscellaneous Facilities: There are two other REXXIMS commands we have not discussed yet. You will need them only in special circumstances:

SET Sets the AIB function parameter or sets the ZZ field on an output message (when you don't want the default value of zero).

STORAGE Obtains or releases storage.

Our sample application does not use these commands.

C.1.3.4 Linking Your REXX Program

IMS expects to find a program with the same name as your PSB (except for a batch program, when the names can be different). This is no different than for other programming languages.

With REXX, this program is a copy of DFSREXX0. We refer to it as the *application stub* to distinguish it from the REXX exec itself.

If you prefer, you can create your stub as an alias of DFSREXX0. This has the advantage of saving DASD space. In our sample application, we created one copy of DFSREXX0 and made all the stubs aliases of that.

DFSREXX0 loads a much larger module DFSREXX1, which creates the REXX environment and calls your REXX exec. Your REXX exec should have the same name as the PSB.

This last point can be quite confusing, as it means there are three things, all with the same name:

- The PSB
- The REXX exec
- The application stub.

If you compile your REXX exec there is no difference. The compiled program is still a member of the REXX source library.

C.1.3.5 Region Considerations

REXX uses different files than other languages. You should add the following DD cards in your region JCL:

SYSEXEC — REXX source library. This may be a concatenation of libraries if you wish. If you have compiled your REXX programs, they still go into this library.

SYSTSPRT — REXX output messages. The SAY command and the IMS trace write to this file.

SYSTSIN — REXX input file. This is where REXX will go to find input after its stack is empty.

In an MPP or IFP region, you should consider preloading

- Your application stubs
- DFSREXX0 (if your stubs are aliases of it)
- DFSREXX1

in order to speed up REXX initialization. Alternatively, you can define the libraries containing these modules to LLA.

In an MPP region you should consider using pseudo WFI as another way of avoiding REXX initialization.

C.1.3.6 Stage One Macros

You define your REXX applications to IMS in exactly the same way as you would for other programming languages.

C.1.4 IMS REXX Trace

IMS provides a trace facility for REXX. There are several levels of trace output, ranging from errors only to the variables before and after every call.

The trace output goes to SYSTSPRT, the same file as REXX SAY commands. This means that your program's messages can be hard to find among the trace output. You can change the trace level during the course of your program. This can be useful if you have isolated a problem to a certain part of your code.

We used the highest level of trace during testing, and switched to the lowest level when we finished testing.

Appendix D. Special Notices

This publication is intended to help system designers and application developers understand the range of options available with IMS Fast Path, for use in new and existing applications. The information in this publication is not intended as the specification of any programming interfaces that are provided by IMS/ESA. See the PUBLICATIONS section of the IBM Programming Announcement for IMS/ESA Version 5 and Version 6 for more information about what publications are considered to be product documentation.

References in this publication to IBM products, programs or services do not imply that IBM intends to make these available in all countries in which IBM operates. Any reference to an IBM product, program, or service is not intended to state or imply that only IBM′s product, program, or service may be used. Any functionally equivalent program that does not infringe any of IBM′s intellectual property rights may be used instead of the IBM product, program or service.

Information in this book was developed in conjunction with use of the equipment specified, and is limited in application to those specific hardware and software products and levels.

IBM may have patents or pending patent applications covering subject matter in this document. The furnishing of this document does not give you any license to these patents. You can send license inquiries, in writing, to the IBM Director of Licensing, IBM Corporation, 500 Columbus Avenue, Thornwood, NY 10594 USA.

Licensees of this program who wish to have information about it for the purpose of enabling: (i) the exchange of information between independently created programs and other programs (including this one) and (ii) the mutual use of the information which has been exchanged, should contact IBM Corporation, Dept. 600A, Mail Drop 1329, Somers, NY 10589 USA.

Such information may be available, subject to appropriate terms and conditions, including in some cases, payment of a fee.

The information contained in this document has not been submitted to any formal IBM test and is distributed AS IS. The use of this information or the implementation of any of these techniques is a customer responsibility and depends on the customer′s ability to evaluate and integrate them into the customer′s operational environment. While each item may have been reviewed by IBM for accuracy in a specific situation, there is no guarantee that the same or similar results will be obtained elsewhere. Customers attempting to adapt these techniques to their own environments do so at their own risk.

The following terms are trademarks of the International Business Machines Corporation in the United States and/or other countries:

AS/400
C/370
CICS/ESA
DProp
IMS
MVS/ESA
OS/390
RACF
BookManager
CICS
DB2
IBM
IMS/ESA
OS/2
Parallel Sysplex
SAA

System/360 System/390
VTAM

The following terms are trademarks of other companies:

C-bus is a trademark of Corollary, Inc.

PC Direct is a trademark of Ziff Communications Company and is used by IBM Corporation under license.

UNIX is a registered trademark in the United States and other countries licensed exclusively through X/Open Company Limited.

Microsoft, Windows, and the Windows 95 logo
are trademarks or registered trademarks of Microsoft Corporation.

Java and HotJava are trademarks of Sun Microsystems, Inc.

Other trademarks are trademarks of their respective companies.

Appendix E. Related Publications

The publications listed in this section are considered particularly suitable for a more detailed discussion of the topics covered in this redbook.

E.1 International Technical Support Organization Publications

For information on ordering these ITSO publications see “How to Get ITSO Redbooks” on page 149.

- *IMS/ESA Version 4 Migration Guide*, GG24-4150.
- *IMS/ESA Version 5 Guide*, GG24-4302.

E.2 Redbooks on CD-ROMs

Redbooks are also available on CD-ROMs. **Order a subscription** and receive updates 2-4 times a year at significant savings.

CD-ROM Title	Subscription Number	Collection Kit Number
System/390 Redbooks Collection	SBOF-7201	SK2T-2177
Networking and Systems Management Redbooks Collection	SBOF-7370	SK2T-6022
Transaction Processing and Data Management Redbook	SBOF-7240	SK2T-8038
AS/400 Redbooks Collection	SBOF-7270	SK2T-2849
RS/6000 Redbooks Collection (HTML, BkMgr)	SBOF-7230	SK2T-8040
RS/6000 Redbooks Collection (PostScript)	SBOF-7205	SK2T-8041
Application Development Redbooks Collection	SBOF-7290	SK2T-8037
Personal Systems Redbooks Collection	SBOF-7250	SK2T-8042

E.3 Other Publications

These publications are also relevant as further information sources:

- *IMS/ESA V5 Administration Guide: Database Manager*, SC26-8012
- *IMS/ESA V5 Administration Guide: System*, SC26-8013
- *IMS/ESA V5 Application Programming: Database Manager*, SC26-8015
- *IMS/ESA V5 Application Programming: Transaction Manager*, SC26-8017
- *IMS/ESA V5 Customization Guide*, SC26-8020
- *IMS/ESA V5 Installation Volume 2: System Definition and Tailoring*, SC26-8024
- *IMS/ESA V5 Utilities Reference: Database Manager*, SC26-8034

How to Get ITSO Redbooks

This section explains how both customers and IBM employees can find out about ITSO redbooks, CD-ROMs, workshops, and residencies. A form for ordering books and CD-ROMs is also provided.

This information was current at the time of publication, but is continually subject to change. The latest information may be found at http://www.redbooks.ibm.com.

How IBM Employees Can Get ITSO Redbooks

Employees may request ITSO deliverables (redbooks, BookManager BOOKs, and CD-ROMs) and information about redbooks, workshops, and residencies in the following ways:

- **PUBORDER** — to order hardcopies in United States
- **GOPHER link to the Internet** - type GOPHER.WTSCPOK.ITSO.IBM.COM
- **Tools disks**

  To get LIST3820s of redbooks, type one of the following commands:

  ```
  TOOLS SENDTO EHONE4 TOOLS2 REDPRINT GET SG24xxxx PACKAGE
  TOOLS SENDTO CANVM2 TOOLS REDPRINT GET SG24xxxx PACKAGE (Canadian users only)
  ```

  To get BookManager BOOKs of redbooks, type the following command:

  ```
  TOOLCAT REDBOOKS
  ```

  To get lists of redbooks, type one of the following commands:

  ```
  TOOLS SENDTO USDIST MKTTOOLS MKTTOOLS GET ITSOCAT TXT
  TOOLS SENDTO USDIST MKTTOOLS MKTTOOLS GET LISTSERV PACKAGE
  ```

  To register for information on workshops, residencies, and redbooks, type the following command:

  ```
  TOOLS SENDTO WTSCPOK TOOLS ZDISK GET ITSOREGI 1996
  ```

  For a list of product area specialists in the ITSO: type the following command:

  ```
  TOOLS SENDTO WTSCPOK TOOLS ZDISK GET ORGCARD PACKAGE
  ```

- **Redbooks Web Site on the World Wide Web**

  http://w3.itso.ibm.com/redbooks

- **IBM Direct Publications Catalog on the World Wide Web**

  http://www.elink.ibmlink.ibm.com/pbl/pbl

  IBM employees may obtain LIST3820s of redbooks from this page.

- **REDBOOKS category on INEWS**
- **Online** — send orders to: USIB6FPL at IBMMAIL or DKIBMBSH at IBMMAIL
- **Internet Listserver**

  With an Internet e-mail address, anyone can subscribe to an IBM Announcement Listserver. To initiate the service, send an e-mail note to announce@webster.ibmlink.ibm.com with the keyword subscribe in the body of the note (leave the subject line blank). A category form and detailed instructions will be sent to you.

Redpieces

For information so current it is still in the process of being written, look at "Redpieces" on the Redbooks Web Site (http://www.redbooks.ibm.com/redpieces.htm). Redpieces are redbooks in progress; not all redbooks become redpieces, and sometimes just a few chapters will be published this way. The intent is to get the information out much quicker than the formal publishing process allows.

How Customers Can Get ITSO Redbooks

Customers may request ITSO deliverables (redbooks, BookManager BOOKs, and CD-ROMs) and information about redbooks, workshops, and residencies in the following ways:

- **Online Orders** — send orders to:

	IBMMAIL	**Internet**
In United States:	usib6fpl at ibmmail	usib6fpl@ibmmail.com
In Canada:	caibmbkz at ibmmail	lmannix@vnet.ibm.com
Outside North America:	dkibmbsh at ibmmail	bookshop@dk.ibm.com

- **Telephone orders**

United States (toll free)	1-800-879-2755
Canada (toll free)	1-800-IBM-4YOU
Outside North America	(long distance charges apply)
(+45) 4810-1320 - Danish	(+45) 4810-1020 - German
(+45) 4810-1420 - Dutch	(+45) 4810-1620 - Italian
(+45) 4810-1540 - English	(+45) 4810-1270 - Norwegian
(+45) 4810-1670 - Finnish	(+45) 4810-1120 - Spanish
(+45) 4810-1220 - French	(+45) 4810-1170 - Swedish

- **Mail Orders** — send orders to:

IBM Publications	IBM Publications	IBM Direct Services
Publications Customer Support	144-4th Avenue, S.W.	Sortemosevej 21
P.O. Box 29570	Calgary, Alberta T2P 3N5	DK-3450 Allerød
Raleigh, NC 27626-0570	Canada	Denmark
USA		

- **Fax** — send orders to:

United States (toll free)	1-800-445-9269
Canada	1-403-267-4455
Outside North America	(+45) 48 14 2207 (long distance charge)

- **1-800-IBM-4FAX (United States)** or **(+1)001-408-256-5422 (Outside USA)** — ask for:

  Index # 4421 Abstracts of new redbooks
  Index # 4422 IBM redbooks
  Index # 4420 Redbooks for last six months

- **Direct Services** - send note to softwareshop@vnet.ibm.com
- **On the World Wide Web**

Redbooks Web Site	http://www.redbooks.ibm.com
IBM Direct Publications Catalog	http://www.elink.ibmlink.ibm.com/pbl/pbl

- **Internet Listserver**

  With an Internet e-mail address, anyone can subscribe to an IBM Announcement Listserver. To initiate the service, send an e-mail note to announce@webster.ibmlink.ibm.com with the keyword subscribe in the body of the note (leave the subject line blank).

Redpieces

For information so current it is still in the process of being written, look at "Redpieces" on the Redbooks Web Site (http://www.redbooks.ibm.com/redpieces.htm). Redpieces are redbooks in progress; not all redbooks become redpieces, and sometimes just a few chapters will be published this way. The intent is to get the information out much quicker than the formal publishing process allows.

IBM Redbook Order Form

Please send me the following:

Title	Order Number	Quantity

First name Last name

Company

Address

City Postal code Country

Telephone number Telefax number VAT number

- Invoice to customer number
- Credit card number

Credit card expiration date Card issued to Signature

We accept American Express, Diners, Eurocard, Master Card, and Visa. Payment by credit card not available in all countries. Signature mandatory for credit card payment.

Glossary

A

Advanced Peer to Peer Communication (APPC). A protocol which programs use to communicate with one another. It means the same as LU6.2.

APPC/MVS. An MVS component that supplies APPC services to programs.

Area. An area is a subset of a DEDB. It is a single data set, although there may be several identical copies of it (see MADS). An area contains all segment types.

B

Base part. A part of a DEDB unit of work. It is the place where IMS stores root segments and their direct dependents.

C

Control Interval (CI). A block of data in a VSAM data set.

Control Interval Update Sequence Number (CUSN). A field which IMS maintains in every CI of a DEDB. It is used to ensure data integrity after an IMS emergency restart in a block-level sharing environment.

Conversation. In APPC, when two programs communicate with each other, they are said to be in "conversation." Every request to start a conversation starts an APPC transaction program.

Commit (or sync) point. The point in time when an application program completes a unit of work. Any changes the program has made to resources can now be made permanent. Any resources the program held can now be released.

Common Programming Interface Communications (CPIC or CPI-C). A programming language for APPC.

Communication Manager/2 (CM/2). A product that supplies APPC and other communication services to OS/2 programs.

D

Database Control (DBCTL). One of the execution modes of IMS. In this mode, IMS manages databases only.

(DMAC). An important DEDB control block.

Data Entry Database (DEDB). A type of IMS database that provides high performance and high availability, including the ability to tolerate permanent I/O errors without disruption.

Dependent overflow. A part of a DEDB unit of work. It is the first place IMS will put database segments that do not fit in the same CI as their root.

See also "Independent Overflow."

Direct Dependent segment (DDEP). An ordinary type of segment within a DEDB. It is a segment that is neither a root nor an SDEP.

E

Expedited Message Handling (EMH). A special IMS process for handling messages faster than normal.

Extended Recovery Facility (XRF). An IMS facility for high availability using a hot-standby alternate system.

Extended Terminal Option (ETO). A function of IMS which allocates terminals as they are needed, rather than requiring you to define them in advance.

F

Fast Path (FP). A group of IMS functions that are simpler but faster than the rest of IMS′s functions. Fast Path comprises DEDBs, MSDBs, and EMH.

Field Search Argument (FSA). A parameter you pass to an IMS field call. It specifies what actions IMS should perform on that field.

Front End Switch (FES). A terminal may enter a message on IMS, but the message processing happens on another system. In this case the IMS system is serving as a Front End Switch.

Full-Function (FF). A collective term for IMS databases that are neither DEDBs nor MSDBs.

H

Hierarchical Direct Access Method (HDAM). A type of IMS database with immediate space reuse and a randomizer to select root positions.

Hierarchical Indexed Direct Access Method (HIDAM). A type of IMS database with immediate space reuse and an index of all root segments. HIDAM stores root segments in key sequence.

High Speed Sequential Processing (HSSP). An especially fast process you may invoke to process a DEDB sequentially in batch.

Highly Parallel Transaction System (HPTS). A new IBM architecture for processing transactions in a sysplex using a large number of loosely coupled computers. IMS Version 5 supports and utilizes the HPTS environment.

I

Independent Overflow (IOVF). A part of a DEDB area. It is the place where IMS puts segments after it has filled the dependent overflow.

L

Logical Unit (LU). A node in the telecommunications network. A logical unit can be a printer, a terminal, a computer, or a program.

Logical Unit 6.1 (LU 6.1). A protocol that programs use to communicate with one another. IMS and CICS use this protocol for Inter System Communication.

Logical Unit 6.2 (LU6.2). A protocol that programs use to communicate with one another. It means the same as APPC.

Lookaside. When IMS looks for a buffer in the buffer pool before reading it from DASD.

M

Main Storage Database (MSDB). A type of IMS database held in main storage.

Mode. A VTAM control block describing the attributes of a network connection.

Multiple Area Data Sets (MADS). An option for a DEDB to have multiple identical copies of its area data sets.

Multiple System Coupling (MSC). A method of communicating between IMS systems. It is independent of the actual connection mechanism.

N

Normal Buffer Allocation (NBA). The number of Fast Path database buffers a region normally uses.

O

Output Thread. An output thread is a separate task which IMS uses to write updated DEDB CIs asynchronously to DASD.

Overflow Buffer Allocation (OBA). The number of extra Fast Path database buffers a region may use above its NBA. IMS does not permit a program to exceed this limit.

P

Partner. In APPC one of the two parties in a conversation.

Parallel Sysplex (PS). A new IBM architecture for running a sysplex of loosely coupled computers. Used by IMS Version 5.

R

Randomizer. A program that uses the key of a segment to decide its position in the database. This is much faster than searching. DEDBs and HDAM databases use randomizers.

The randomizer does not select a arbitrary position for the segment; it chooses one of the predefined root anchor points.

Relative Byte Address (RBA). A position within a data set, measured in bytes from the start. The first byte has an RBA of zero, the second byte an RBA of one (because it is one byte from the start), and so on.

Root Addressable part. A part of a DEDB area. It is the collective term for the base and dependent overflow parts of a DEDB.

Root Anchor Point (RAP). A special kind of pointer in a DEDB or HDAM database. A RAP points to a root segment. The randomizer decides which RAP points to which root segment.

S

Side information. (1) A set of information about a particular APPC program. (2) The name of this set of information is known.

Sequential Dependent part. A part of a DEDB area. It is the place where IMS stores SDEP segments.

Sequential Dependent segment (SDEP). A special type of segment within a DEDB. It is fast to insert but slow to read.

Subset pointer (SSP). A special kind of database pointer which an application may set and use. Only DEDBs support subset pointers.

Sync point. Another name for commit point.

Synonym chaining. A phenomenon of DEDBs and HDAM databases. These databases use a randomizer to select the position of a root in the database. This position is known as a root anchor point (RAP).

Sometimes the randomizer places more than one root at the same RAP. These two roots are then said to be synonyms of one another. IMS maintains chain of pointers from one synonym to the next. This is called a synonym chain. IMS maintains the chain in key sequence. The RAP points to the first root in the chain.

How often this happens depends on how many roots you have, how many RAPs you have and on the randomizer logic.

See “randomizer” and “Root Anchor Point.”

T

Transaction Program (TP). Another name for an APPC application program.

U

unit of work. (1) The smallest amount of application processing that is internally consistent. All elements of a unit of work are committed or backed out together. (2) In a DEDB, a unit of work is a predetermined number of CIs.

V

Virtual Storage Constraint Relief (VSCR). Certain areas of Virtual Storage are in short supply. Successive releases of IMS and other products have sought to reduce their use of these areas of Virtual Storage. This is known as “Virtual Storage Constraint Relief.”

Virtual Storage Option (VSO). A type of DEDB that is held in a data space.

W

Wait For Input (WFI). A type of IMS program that remains idle between messages.

List of Abbreviations

ACB	application control block
ACBGEN	application control block generation
ACF	advanced communications facility
ACK	acknowledgement
ADS	area data set
AIB	application interface block
API	application program interface
APPC	advanced program-to-program communication
APPC/MVS	advanced program-to-program communication/multiple virtual storage (IBM)
APPLCTN	application
ASCII	American National Standard Code for Information Interchange
BMP	batch message processing, region (IMS)
BTS	batch terminal simulator
CHNG	change
CI	control interval
CIC	concurrent image copy
CICS	customer information control system (IBM)
CICS/ESA	customer information control system/enterprise systems architecture (IBM)
CNT	communication name table
CNTL	control
COBOL	common business oriented language
COS	class of service
CPI	common programming interface (SAA)
CPI-CI	common programming interface for communications (SAA)
CPIC	common programming interface for communications (SAA)
CPU	central processing unit
CSA	common storage area
DASD	direct access storage device
DB	data base
DB/DC	data base/data communications
DBCTL	Data Base ControL Subsystem
DBD	data base description
DBDGEN	data base description generation
DBR	data base recovery
DBRC	data base recovery control (IMS)
DD	data set definition
DEDB	data entry data base
DEQ	dequeue
DFP	data facility product (MVS)
DL/I	data language 1
DLET	delete
DLI	data language interface
DMAC	data management area control block
DPROP	data propagator (IBM program product, note - case should be DProp)
DRA	database resource adapter
EBCDIC	extended binary coded decimal interchange code
ECSA	extended common service area
EMH	expedited message handling
EOV	end of volume
ERE	emergency restart
ESDS	entry sequenced data set (VSAM)
ESTAE	extended subtask ABEND exit
ETO	Extended Terminal Option (IMS DC)
FIFO	first in/first out
FLD	field
FP	Fast Path
GHN	get hold next (IMS)
GHNP	get hold next within parent (IMS)
GHU	get hold unique (IMS)

GN	get next (IMS)
GNP	get next within parent (IMS)
GSAM	Generalized sequential access method
GU	get unique (IMS)
HDAM	hierarchic direct access method
HIDAM	hierarchic indexed direct access method
HISAM	hierarchic indexed sequential access method
HPTS	high-performance transaction system
HSAM	hierarchic sequential access method
I/O	input/output
IBM	International Business Machines Corporation
IC	image copy
ICIP	improved control interval processing (VSAM)
ID	identification/identifier
IDCAMS	the program name for access method services (OP SYS)
IEBCOPY	utility program (MVS)
IEBGENER	utility program (MVS)
IFP	IMS Fast Path
IMS	Information Management System
IMS/ESA	Information Management System/Enterprise Systems Architecture
IMS/VS	Information Management System/Virtual Storage
IMSGEN	IMS/VS System Generation
INIT	initialize/initial/initiate
IOPCB	input/output program communication block
IRLM	internal resource lock manager
ISC	intersystem communications
ISRT	insert (IMS)
ITSO	International Technical Support Organization
JCL	job control language (MVS and VSE)
LIFO	last in/first out
LLA	library lookaside (MVS/ESA)
LTERM	logical terminal
LU	logical unit
	logon work area (TSO)
MADS	multiple area data set
MB	megabyte, 1,000,000 bytes (1,048,576 bytes memory)
MFS	message format service (IMS/DC screen mapper)
MPP	message processing program
MSC	multiple systems coupling
MSDB	main storage data base
MVS	Multiple Virtual Storage (IBM System 370 & 390)
MVS/ESA	Multiple Virtual Storage/Enterprise Systems Architecture (IBM)
NBA	normal buffer allocation
NCP	network control program
NRE	normal restart
OLDS	on-line log data set
OSAM	overflow sequential access method
PASCAL	a programming language
PCB	program communication block
PLC	processing limit count
POS	position
PROCLIB	Procedure Library (IBM System/360)
PROCOPT	processing option
PSB	program specification block
PSBGEN	program specification block generation
PTF	physical twin forward (IMS)
PTF	program temporary fix
PURG	purge
RACF	resource access control facility
RAP	root anchor point (HDAM)
RECON	recovery control (data set)
REPL	replace
REXX	restructured extended executor language
RLDS	recovery log data set
ROLB	an IMS facility to undo the effect of previous updates
RSR	remote site recovery

SDUMP	SVC dump parameter list (MVS control block)
SENSEG	sensitive segment
SETO	set option
SETR	set range (HSSP)
SHISAM	simple hierarchic indexed sequential access method
SLUP	secondary logical unit program (IMS)
SMB	scheduler message block
SNA	Systems Network Architecture (IBM)
SPE	small programming enhancement
SRB	service request block (MVS control block)
SSA	segment search argument (IMS)
SSP	subset pointer
STA	start command
STO	stop command
SYNC	synchronize
SYSDEF	system definition (frame)
TCO	time-controlled operations
TIIF	time-initiated input facility
TM	transaction manager
TP	transaction program/process (OSI)
TSO	time sharing option
UNIX	an operating system developed at Bell Laboratories (trademark of UNIX System Laboratories, licensed exclusively by X/Open C
UOW	unit of work
VR	Virtual Route (SNA)
VSAM	Virtual Storage Access Method (IBM)
VSCR	virtual storage constraint relief
VSO	virtual storage option
VTAM	Virtual Telecommunications Access Method (IBM)
WADS	write ahead data set
WTL	write to log
WTO	write to operator
WTOR	write to operator with reply
WTP	write to programmer
XRF	extended recovery facility

Index

D

E

N

O

P

R

S

T

U

V

W

X

ITSO Redbook Evaluation

IMS Fast Path Solutions Guide
SG24-4301-00

Your feedback is very important to help us maintain the quality of ITSO redbooks. **Please complete this questionnaire and return it using one of the following methods:**

- Use the online evaluation form found at http://www.redbooks.com
- Fax this form to: USA International Access Code + 1 914 432 8264
- Send your comments in an Internet note to redbook@vnet.ibm.com

Please rate your overall satisfaction with this book using the scale:
(1 = very good, 2 = good, 3 = average, 4 = poor, 5 = very poor)

Overall Satisfaction _____

Please answer the following questions:

Was this redbook published in time for your needs? Yes____ No____

If no, please explain:

_____

_____

_____

_____

What other redbooks would you like to see published?

_____

_____

_____

Comments/Suggestions: **(THANK YOU FOR YOUR FEEDBACK!)**

_____

_____

_____

_____

_____

Printed in U.S.A.