IMS Application Development Facility II
Version 2 Release 2

Application Development Guide

Publication Number
SH20-6595-01

File Number
S/370/4300-32

Program Number
5665-348

Second Edition (June 1986)

This edition applies to Version 2, Release 2 of the program product IMS Application Development Facility II (5665-348), and to all subsequent releases and modifications unless otherwise indicated in new editions or Technical Newsletters.

Information in this publication is subject to change. Changes will be published in new editions or technical newsletters. Before using this publication, consult either your *IBM System/370 and 4300 Processors Bibliography* (GC20-0001) or *IBM System/370 and 4300 Processors Bibliography of Industry Systems Systems and Application Programs* (GC20-0370) to learn which editions and technical newsletters are current and applicable.

References in this publication to IBM products, programs, or services do not imply that IBM intends to make these available outside the United States. Any reference to an IBM program product in this document is not intended to state or imply that only IBM's program products may be used. Any functionally equivalent program products may be used instead.

Requests for copies of IBM publications should be made to your IBM representative or to the IBM branch office serving your locality.

A form for readers' comments has been provided at the back of this publication. If this form has been removed, address comments to:

IBM Corporation
Information Processing
Department 6DD
220 Las Colinas Blvd.
Irving, Texas 75039-5513

IBM may use or distribute any of the information you supply in any way it believes appropriate without incurring any obligation whatever. You may, of course, continue to use the information you supply.

PREFACE

This manual is a procedural guide for the planning and development of IMS Application Development Facility II Version 2 Release 2 applications.

This manual consists of thirteen chapters and five appendixes.

- **Chapter 1, "IMSADF II Concepts and Overview"** describes the end-user perspective of IMSADF II.
- **Chapter 2, "Static Rules and the Rules Generator"** contains a brief list of Rules Generator statements and some major keywords.
- **Chapter 3, "Sign-On Security"** contains information pertaining to security in the end-user environment.
- **Chapter 4, "The Auditor and the Audit Data Base"** describes the flow of control in an IMSADF II application and how a developer can specify the logic for data validation and manipulation.
- **Chapter 5, "Message Sending and Display"** contains information about messages to and from end-users.
- **Chapter 6, "Complex Transactions"** provides assistance and examples of complex application processing.
- **Chapter 7, "Secondary Transactions and IMS/VS Message Routing"** describes how an IMSADF II transaction can cause another transaction to be invoked.
- **Chapter 8, "Special Processing"** explains how IMSADF II standard processing can be extended.
- **Chapter 9, "Exits"** describes how the Auditor functions in Chapter 4 can be extended.
- **Chapter 10, "Batch Processing"** contains information regarding IMSADF II batch applications.
- **Chapter 11, "Nonconversational Processing"** provides assistance in developing nonconversational applications.
- **Chapter 12, "HELP Facility"** describes how a developer can provide online HELP for the end-user.
- **Chapter 13, "National Language Support"** contains a description of the National Language Support provided by IMSADF II.
- The appendixes include:

  Appendix A, "Sample System Rules Generator Statements"

  Appendix B, "Alternate Twin Processing Techniques"

  Appendix C, "Report Writing Example"

  Appendix D, "Application Implementation"

  Appendix E, "Switching Between COBOL and IMSADF II Transactions"

RELATED PUBLICATIONS

IMSADF II PUBLICATIONS

- IMS Application Development Facility II Version 2 Release 2 General Information, GH20-6591.
- IMS Application Development Facility II Version 2 Release 2 User Reference, SH20-6592.
- IMS Application Development Facility II Version 2 Release 2 Installation Guide, SH20-6593.
- IMS Application Development Facility II Version 2 Release 2 Application Development Reference, SH20-6594.
- IMS Application Development Facility II Version 2 Release 2 Application Development Guide, SH20-6595.
- IMS Application Development Facility II Version 2 Release 2 Rules Documentation User's Guide, SH20-6596.
- IMS Application Development Facility II Version 2 Release 2 Data Dictionary Extension User's Guide, SH20-6597.
- IMS Application Development Facility II Version 2 Release 2 Master Index, SH20-6599.
- IMS Application Development Facility II Version 2 Release 2 Introduction to Using the Interactive ADF, SH20-6601.
- IMS Application Development Facility II Version 2 Release 2 Interactive ADF Administration Guida, SH20-6602.
- IMS Application Development Facility II Version 2 Release 2 DATABASE 2 Application Specification Guide, SH20-6603.
- IMS Application Development Facility II Version 2 Release 2 Diagnosis Guide, LY20-6401.

OTHER PUBLICATIONS

- Information Management System/ Virtual Storage (IMS/VS) General Information Manual, GH20-1260
- CICS General Information Manual, GC33-0155

CONTENTS

FIGURES

CHAPTER 1. IMSADF II CONCEPTS AND OVERVIEW

Using a standard IMSADF II conversational transaction driver, the simplest and most commonly used, the end-user can step through a series of displays, answering fill-in-the-blank questions and selecting items from tailored menus. The sequence of menus and displays is shown in Figure 1-1. After the user signs on and passes the security clearance, option menus are displayed to allow selection of functions and transactions. If necessary, the user is prompted to enter the key information needed to retrieve the data to be displayed.

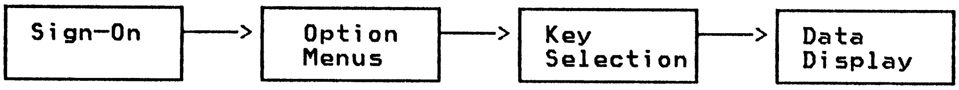
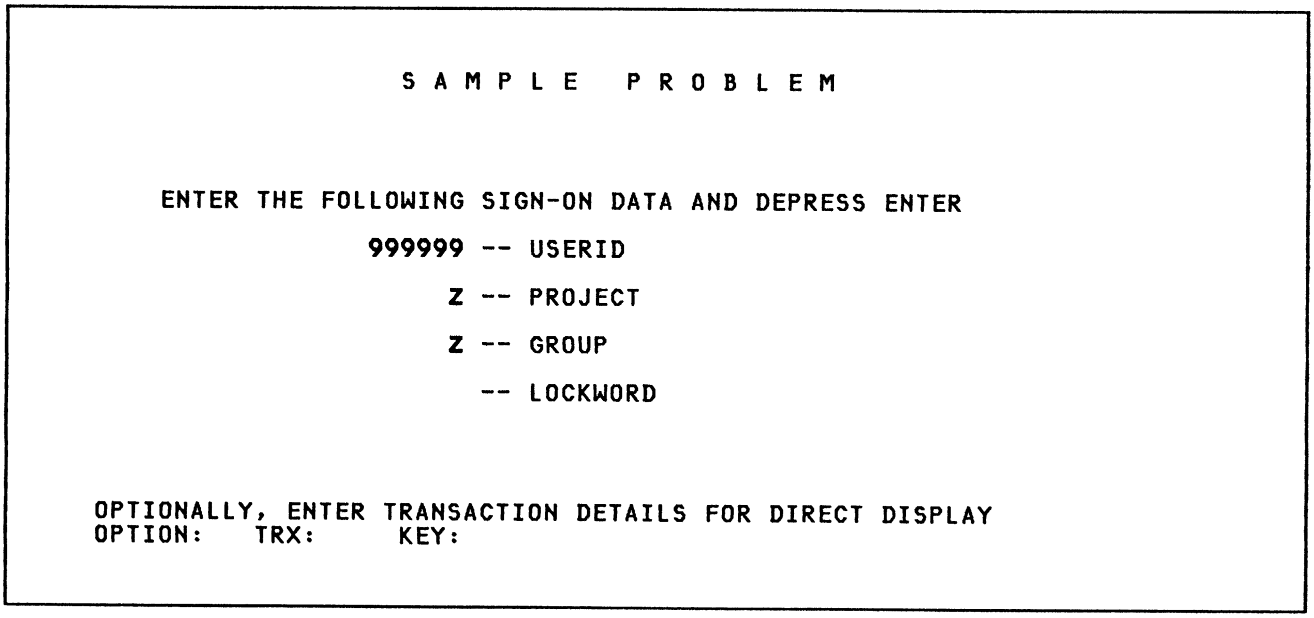


Figure 1-1. IMSADF II Standard Screen Sequence (Conversational)

THE TERMINAL USER'S VIEW

You begin IMSADF II conversational processing by displaying the IMSADF II Sign-On screen as shown in Figure 1-2 where you enter your user ID and project/group code.



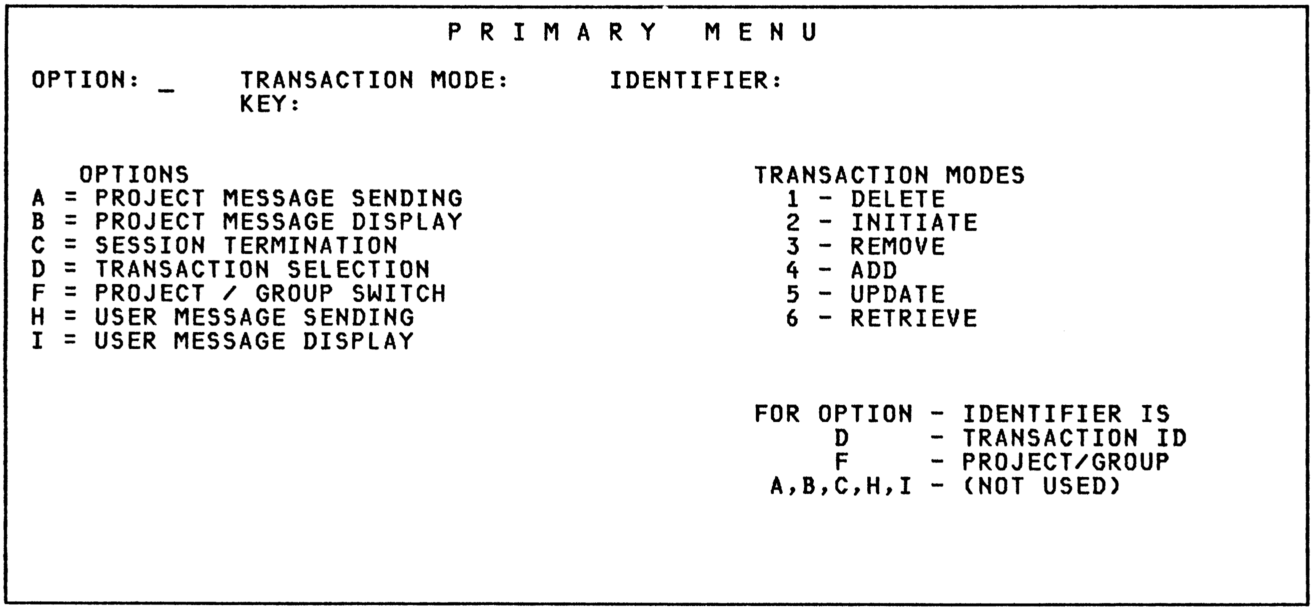
```
                     S A M P L E   P R O B L E M


      ENTER THE FOLLOWING SIGN-ON DATA AND DEPRESS ENTER
                   999999 -- USERID
                        Z -- PROJECT
                        Z -- GROUP
                          -- LOCKWORD



    OPTIONALLY, ENTER TRANSACTION DETAILS FOR DIRECT DISPLAY
    OPTION:   TRX:     KEY:
```

Figure 1-2. Sign-On Screen

IMSADF II checks your security profile to make sure you are authorized to use this application system and to verify the mode (display, update, insert, delete) that is permitted against available transactions. The **lockword** (IMSADF II's term for password) will be checked by an installation-defined exit, if implemented.

OPTION MENUS

Next, the Primary Option Menu, which lists functions available to the user of this application system, appears (Figure 1-3).



```
                              P R I M A R Y   M E N U

 OPTION: _    TRANSACTION MODE:      IDENTIFIER:
              KEY:


    OPTIONS                                      TRANSACTION MODES
 A = PROJECT MESSAGE SENDING                       1 - DELETE
 B = PROJECT MESSAGE DISPLAY                       2 - INITIATE
 C = SESSION TERMINATION                           3 - REMOVE
 D = TRANSACTION SELECTION                         4 - ADD
 F = PROJECT / GROUP SWITCH                        5 - UPDATE
 H = USER MESSAGE SENDING                          6 - RETRIEVE
 I = USER MESSAGE DISPLAY


                                                 FOR OPTION - IDENTIFIER IS
                                                      D     - TRANSACTION ID
                                                      F     - PROJECT/GROUP
                                                  A,B,C,H,I - (NOT USED)
```

Figure 1-3. Primary Option Menu Screen

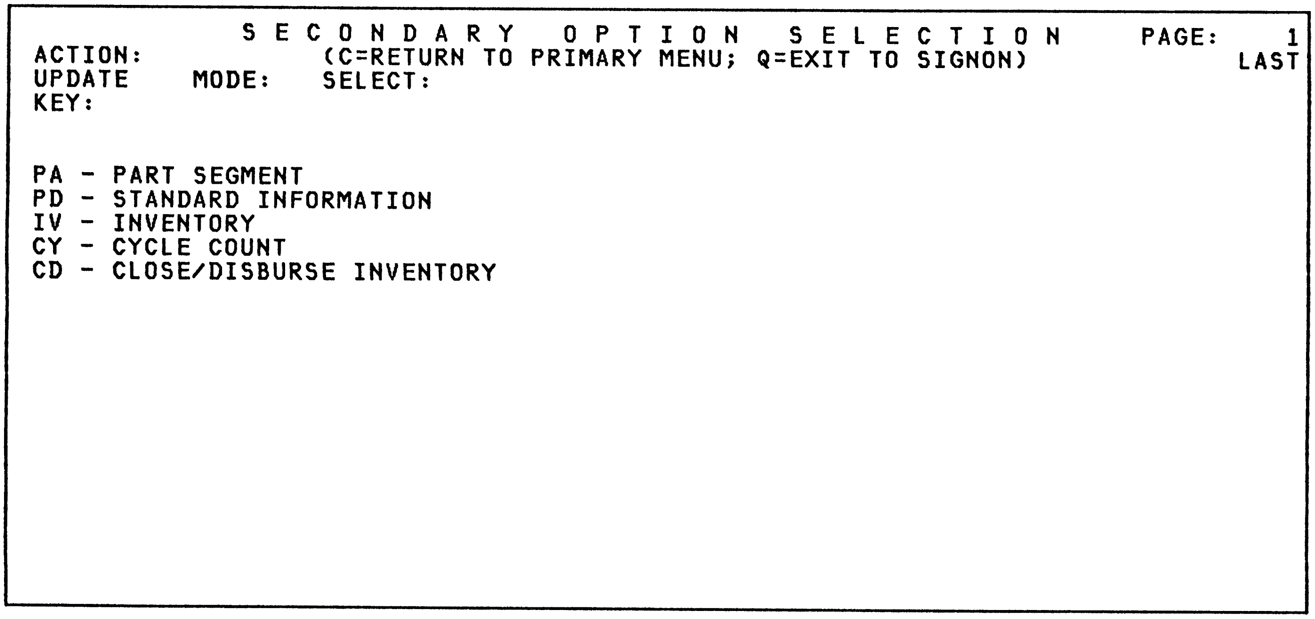
Function options A, B, H, and I, if they are available in this application system, allow the user to send messages to and receive messages from other users or project/groups.

If option F is available, the user may switch to another project/group code, with a different security profile, without logging off and signing back on to IMSADF II. This feature exists because, although a project/group may use only one application system, a user may belong to more than one project/group.

Option C is used to terminate this IMSADF II session.

The main option is D - Transaction Selection. If the user enters option D, he must also select a transaction mode to indicate what is to be done against the transaction. IMSADF II will check the user's authorization to use the transaction mode chosen. The user's security profile will list the lowest mode he is allowed to use for each transaction; that is, if a user is authorized to use transaction mode 3, he may also use modes 4, 5, and 6.

If the two-character transaction ID is not known, the user can press ENTER and receive a display of all transactions available in this system with which he may work. This display, shown in Figure 1-4, is the Secondary Option Menu.



```
               S E C O N D A R Y   O P T I O N   S E L E C T I O N     PAGE:    1
ACTION:             (C=RETURN TO PRIMARY MENU; Q=EXIT TO SIGNON)                LAST
UPDATE    MODE:     SELECT:
KEY:


PA - PART SEGMENT
PD - STANDARD INFORMATION
IV - INVENTORY
CY - CYCLE COUNT
CD - CLOSE/DISBURSE INVENTORY
```

Figure 1-4. Secondary Option Menu Screen

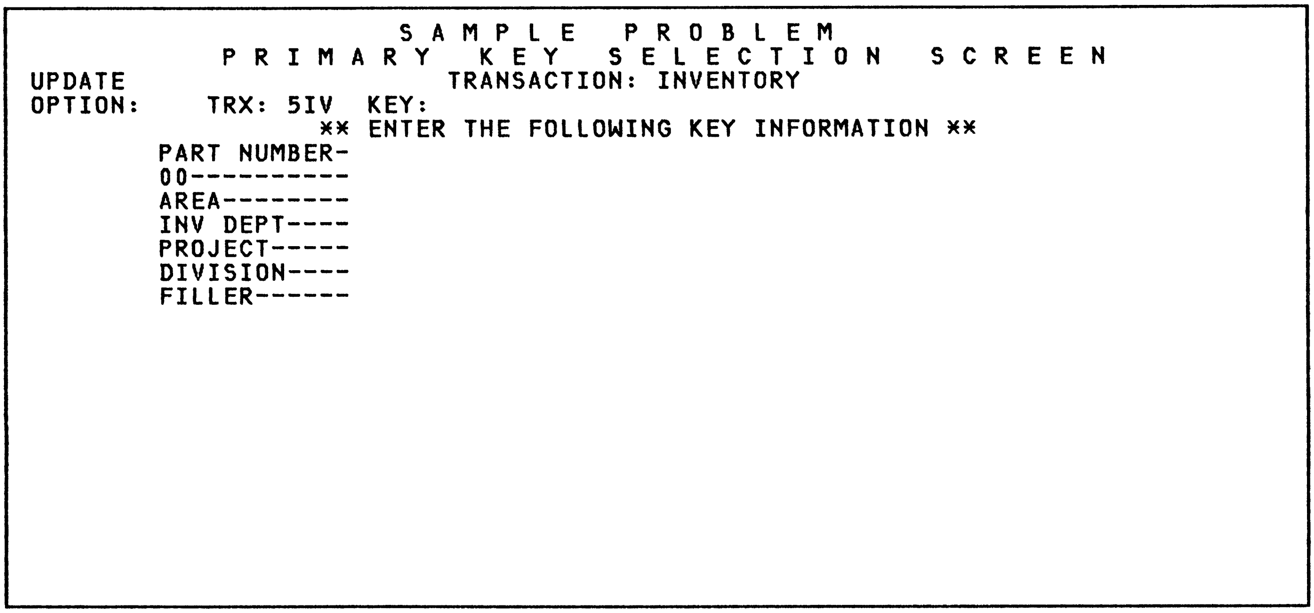
If the user had entered the transaction ID on the Primary Option Menu screen, the Secondary Option Menu screen would not have been displayed. On either of these option menus, the user can enter a key as well. The transaction uses this key to retrieve segments from the data bases. The key is the concatenation of all the keys required to retrieve a segment to be used in the transaction. Segment retrieval can also be carried out under control of dynamic rules.

The user can also enter the OPTION, TRANSACTION MODE and ID, and KEY on the Sign-On screen to bypass all menus and go directly to the Data Display screen.

KEY SELECTION

If the user is unable to enter the necessary concatenated key directly, a function known as key selection is available. This function checks the concatenated key, if one has been entered, and looks in the data base for the segment occurrences specified. If the user has entered no keys or has entered keys that are incomplete or inconsistent with what is on the data base, IMSADF II will prompt the user to enter the correct information.

First, the user is presented with a formatted request for the keys that make up the concatenated key: the Primary Key Selection screen, Figure 1-5.



```
                         S A M P L E   P R O B L E M
               P R I M A R Y   K E Y   S E L E C T I O N   S C R E E N
UPDATE                     TRANSACTION: INVENTORY
OPTION:    TRX: 5IV  KEY:
                  ** ENTER THE FOLLOWING KEY INFORMATION **
        PART NUMBER-
        00----------
        AREA--------
        INV DEPT----
        PROJECT-----
        DIVISION----
        FILLER------
```

Figure 1-5. Primary Key Selection Screen

In this example, the segments each have two-character IDs that happen to coincide with four of the transaction IDs in the application system. This convention has a special significance: each of these four transactions allows display and update of the correspondingly named segment, plus all those above it in the hierarchical path (see Figure 1-6).

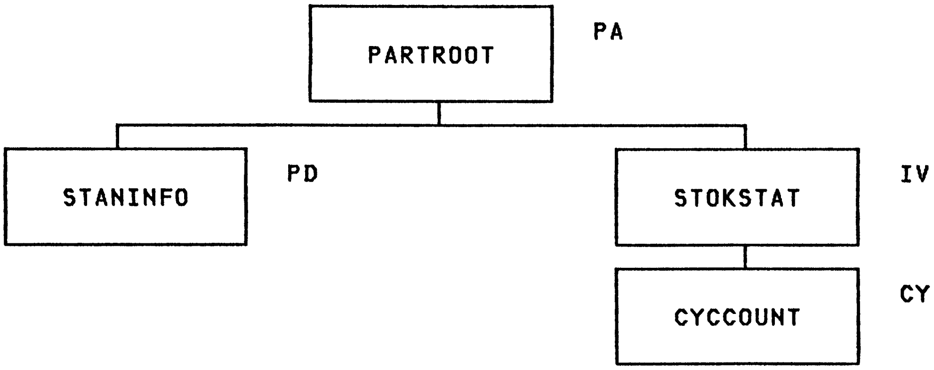
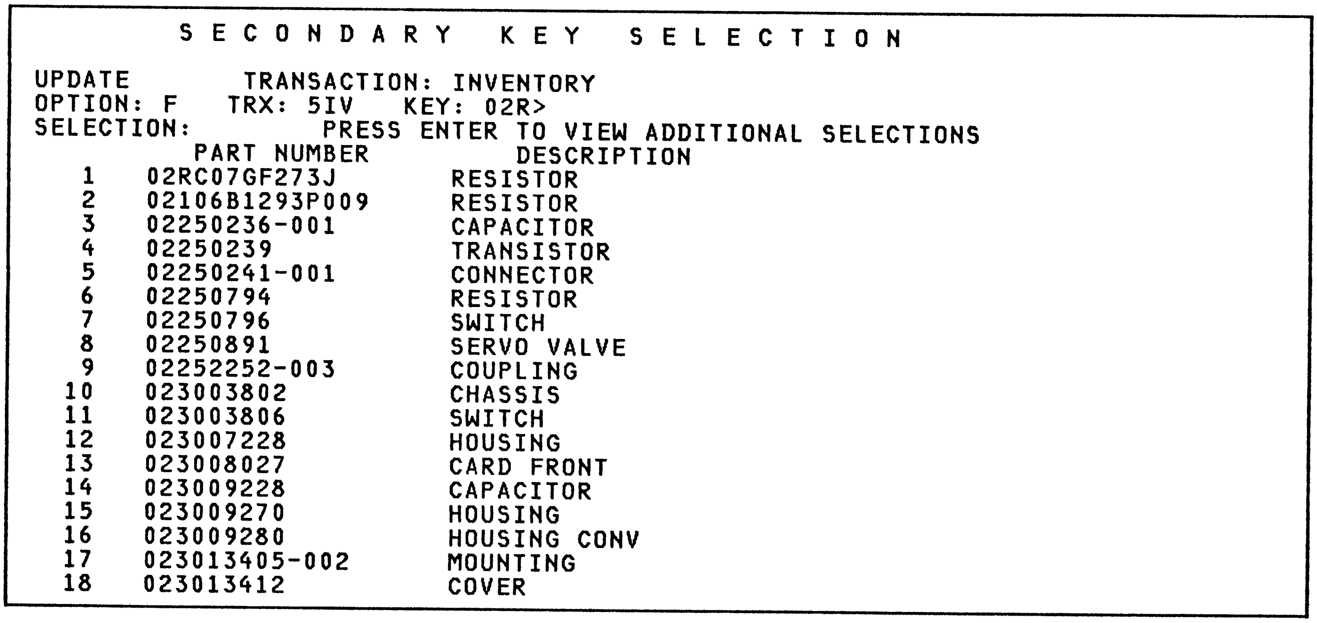


Figure 1-6. Sample Data Base

Thus, the CY transaction prompts the user for the keys of the PA, IV, and CY segments in order to retrieve them for display and update, but the PA transaction deals only with the root segment. These four transactions provide a data base maintenance application.

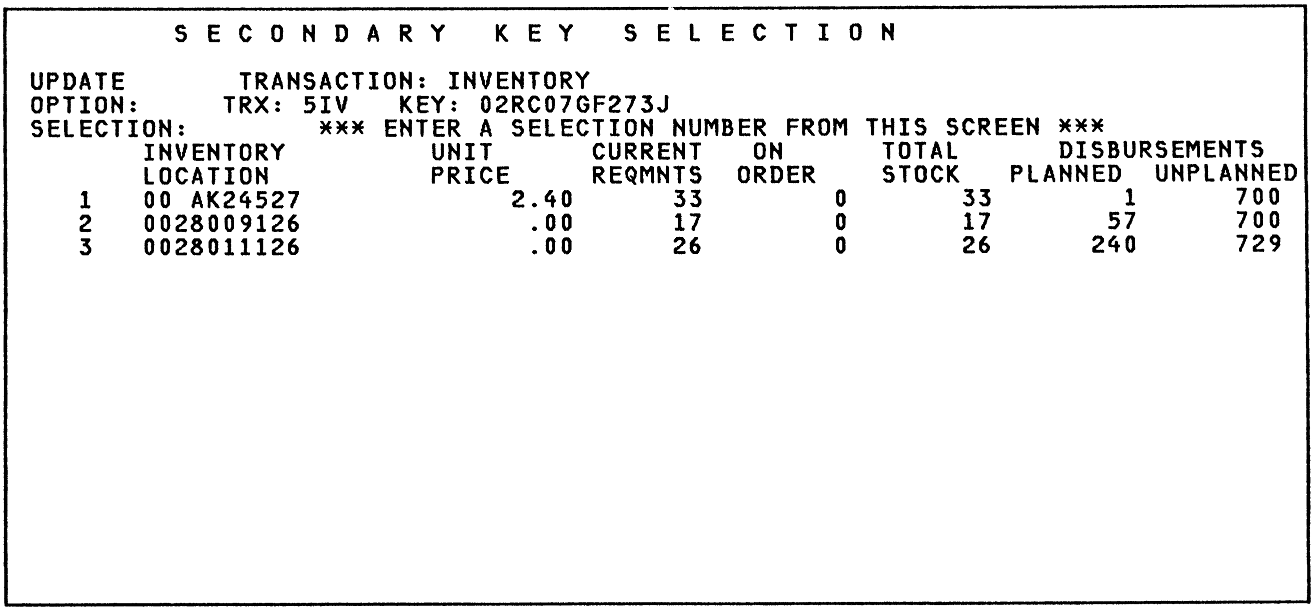
The lowest level segment retrieved by a transaction in a hierarchical path is known as the **target segment**. When a transaction involves multiple hierarchical paths in one or more data bases, it is said to have multiple target segments -- one for each path. When defining a transaction, you will name the target segments. At execution time, then, the user will be prompted to enter target segment keys, as well as those of higher level segments, with a screen similar to that shown in Figure 1-5.

If the user still cannot enter some or all of the keys, IMSADF II provides a data base browsing capability as a standard part of the application. You can restrict or eliminate the browsing capability when you develop a transaction. If you include the browsing feature in a transaction, IMSADF II will provide a Secondary Key Selection screen for this purpose. Figure 1-7 and Figure 1-8 illustrate how browsing can be allowed at different levels in the data base down to the target segment. A transaction supporting many target segments can provide browsing at every level in every path down to the target.



```
          S E C O N D A R Y   K E Y   S E L E C T I O N

UPDATE       TRANSACTION: INVENTORY
OPTION: F   TRX: 5IV   KEY: 02R>
SELECTION:       PRESS ENTER TO VIEW ADDITIONAL SELECTIONS
          PART NUMBER         DESCRIPTION
   1    02RC07GF273J        RESISTOR
   2    02106B1293P009      RESISTOR
   3    02250236-001        CAPACITOR
   4    02250239            TRANSISTOR
   5    02250241-001        CONNECTOR
   6    02250794            RESISTOR
   7    02250796            SWITCH
   8    02250891            SERVO VALVE
   9    02252252-003        COUPLING
  10    023003802           CHASSIS
  11    023003806           SWITCH
  12    023007228           HOUSING
  13    023008027           CARD FRONT
  14    023009228           CAPACITOR
  15    023009270           HOUSING
  16    023009280           HOUSING CONV
  17    023013405-002       MOUNTING
  18    023013412           COVER
```

Figure 1-7. Secondary Key Selection Screen (Data Base Browsing)

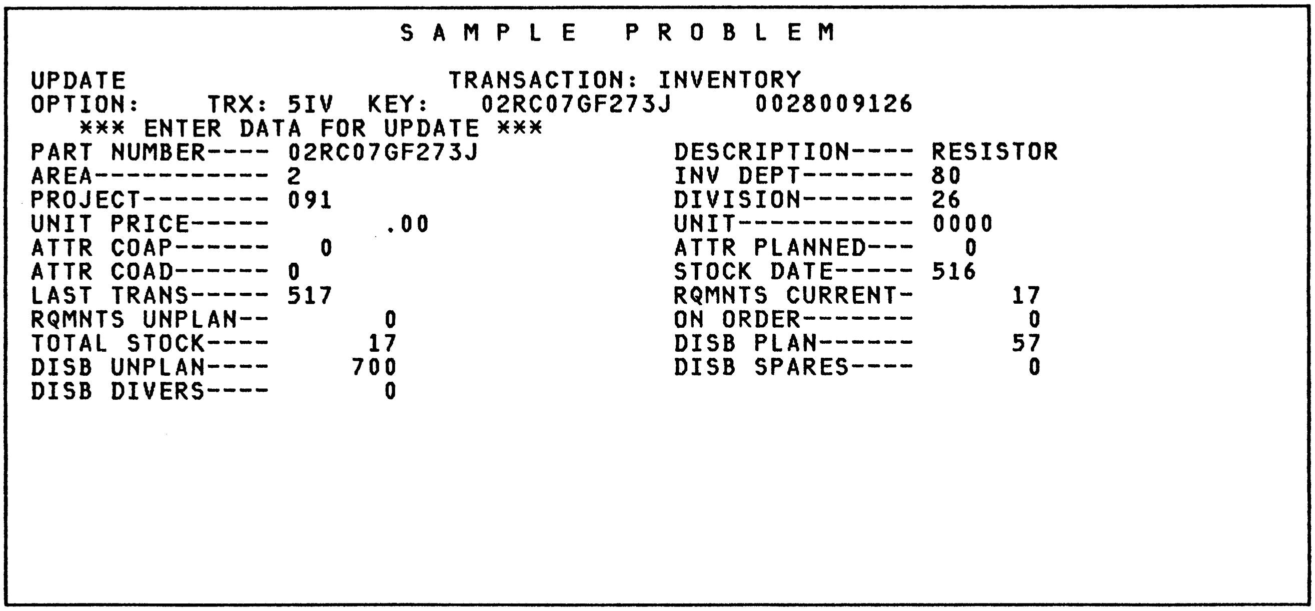


```
          S E C O N D A R Y   K E Y   S E L E C T I O N

UPDATE       TRANSACTION: INVENTORY
OPTION:     TRX: 5IV   KEY: 02RC07GF273J
SELECTION:        *** ENTER A SELECTION NUMBER FROM THIS SCREEN ***
       INVENTORY            UNIT      CURRENT   ON       TOTAL       DISBURSEMENTS
       LOCATION             PRICE     REQMNTS  ORDER     STOCK    PLANNED  UNPLANNED
   1   00 AK24527              2.40        33        0        33        1        700
   2   0028009126               .00        17        0        17       57        700
   3   0028011126               .00        26        0        26      240        729
```

Figure 1-8. Browsing at a Lower Level in the Data Base

DATA DISPLAY AND UPDATE

Finally the user sees the data he has chosen on a Data Display screen (Figure 1-9).



```
                            S A M P L E   P R O B L E M

UPDATE                          TRANSACTION: INVENTORY
OPTION:    TRX: 5IV  KEY:   02RC07GF273J     0028009126
    *** ENTER DATA FOR UPDATE ***
PART NUMBER---- 02RC07GF273J          DESCRIPTION---- RESISTOR
AREA----------- 2                     INV DEPT------- 80
PROJECT-------- 091                   DIVISION------- 26
UNIT PRICE-----       .00             UNIT----------- 0000
ATTR COAP------   0                   ATTR PLANNED---   0
ATTR COAD------ 0                     STOCK DATE----- 516
LAST TRANS----- 517                   RQMNTS CURRENT-      17
RQMNTS UNPLAN--      0                ON ORDER-------       0
TOTAL STOCK----     17                DISB PLAN------      57
DISB UNPLAN----    700                DISB SPARES----       0
DISB DIVERS----      0
```

Figure 1-9. Data Display Screen

This is where changes are made to the data, if appropriate.

The user can also go directly to the Data Display screen from any of the previous screens (Primary Option Menu, Secondary Option Menu, Primary Key Selection, or Sign-On) simply by entering all the necessary information: the function option, transaction ID and mode, and concatenated key.

The user may switch directly from this Data Display screen to the Data Display screen for another transaction simply by entering a new transaction ID and mode (next to **TRX** on the Data Display screen) and/or a new concatenated key (next to **KEY**).

If the user is working with multiple target segments in a multiple-path transaction, he may enter **N** on this screen (next to **OPTION**) to work sequentially through the various target segments.

Finally, you can develop the transaction so that the user may display other segments in a transaction simply by entering new keys over the displayed key fields (e.g., Part Number).

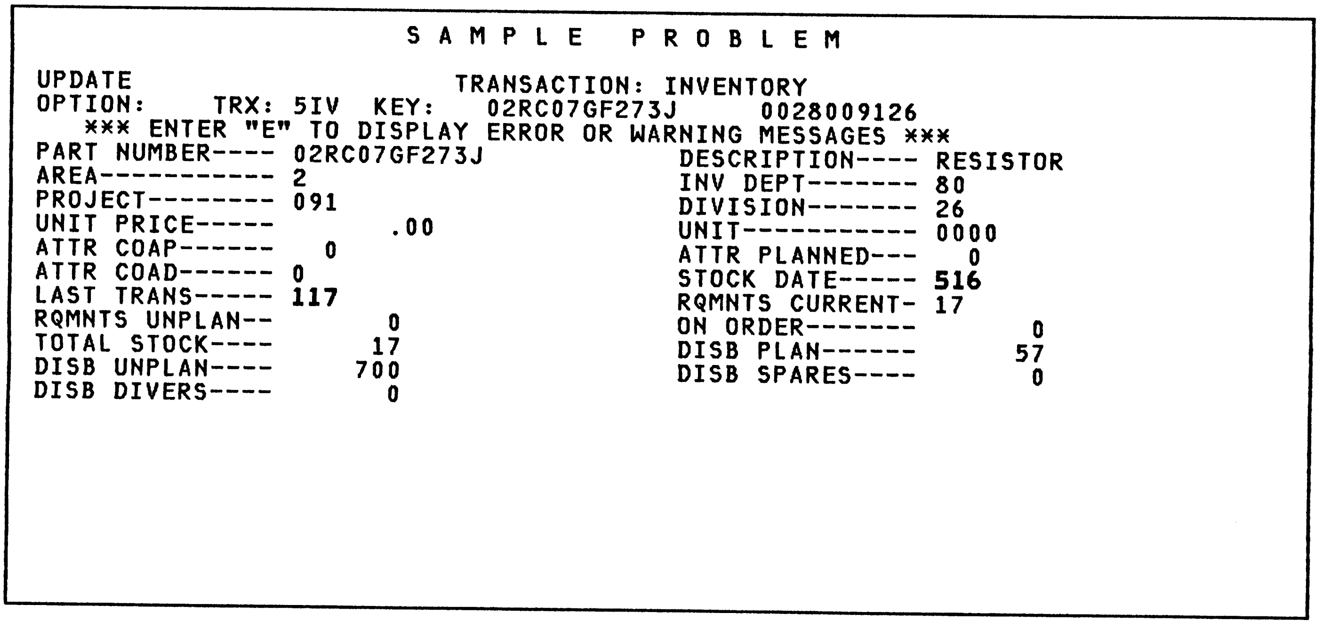
Transaction Modes

The transaction mode selected on the Primary Option Menu screen controls what the user can do with the data seen on the Data Display screen.

Transaction mode 6, Retrieve, allows the user to look at the data; it cannot be altered.

Transaction mode 5, Update, allows the user to update data and place the new information in the data base. However, certain data fields may be defined as display only. Other data fields may be modifiable, but subject to validation and other processing. The dynamic rules you develop for the transaction will define which fields can be modified and under what conditions.

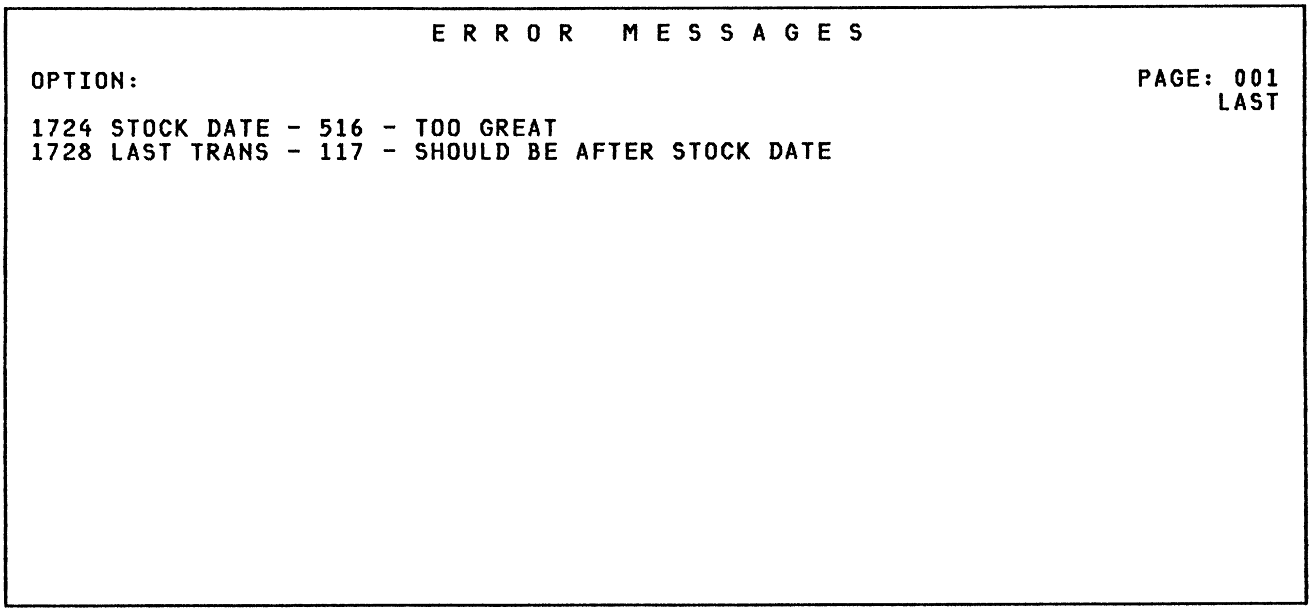
IMSADF II will not allow a user to modify a field that is defined as display only. If a user tries to place invalid values in a field, IMSADF II will display a message and highlight the fields in error (see Figure 1-10).



```
                         S A M P L E   P R O B L E M

UPDATE                             TRANSACTION: INVENTORY
OPTION:     TRX: 5IV  KEY:    02RC07GF273J       0028009126
   *** ENTER "E" TO DISPLAY ERROR OR WARNING MESSAGES ***
PART NUMBER---- 02RC07GF273J            DESCRIPTION---- RESISTOR
AREA----------- 2                       INV DEPT------- 80
PROJECT-------- 091                     DIVISION------- 26
UNIT PRICE-----       .00               UNIT----------- 0000
ATTR COAP------   0                     ATTR PLANNED---   0
ATTR COAD------ 0                       STOCK DATE----- 516
LAST TRANS----- 117                     RQMNTS CURRENT- 17
RQMNTS UNPLAN--     0                   ON ORDER-------       0
TOTAL STOCK----    17                   DISB PLAN------      57
DISB UNPLAN----   700                   DISB SPARES----       0
DISB DIVERS----     0
```

Figure 1-10. Data Display Screen with Error Notification

If the user needs more information to correct the field, he may enter **E** in the OPTION field to display additional error messages (Figure 1-11).



```
                      E R R O R   M E S S A G E S
 OPTION:                                                          PAGE: 001
                                                                       LAST
 1724 STOCK DATE - 516 - TOO GREAT
 1728 LAST TRANS - 117 - SHOULD BE AFTER STOCK DATE
```

Figure 1-11. Error Message Screen

The user then presses ENTER to return to the Data Display screen to complete the updates.

Transaction mode 4, Add, is used mainly with transactions that have a single target segment. Unless you define the transaction otherwise, the user can insert only the target segment and amend the higher level segments using this transaction mode. Generally, combinations of insertions and amendments to segments in one or several paths are handled with transaction mode 5 (Update).

Transaction mode 3, Remove, is used for deleting segments, but again is used mainly on transactions that have a single target segment with an ID equal to the transaction ID.

Transaction modes 1 and 2 are interchangeable with modes 3 and 4, respectively. It is recommended that modes 3 and 4 be used rather than 1 and 2.

The Data Display screens shown in this section are in the default format produced by the Rules Generator. You can use static rules to customize the format of these screens when you develop a transaction.

MENU SEQUENCE AND ADDITIONAL PROCESSING

By simply adding to and subtracting from the IMSADF II standard application architecture, you can very quickly and easily develop a wide variety of transactions.

The sequence of menus and possible variations in that sequence are summarized in Figure 1-12. The loop shown in Secondary Key Selection indicates the possibility of browsing through several different levels or paths in one or more data bases. The user can return to the Primary Option Menu from any of the screens by entering OPTION C. He can return to the Sign-On screen by entering OPTION Q.

The symbol X marks the points at which the user can go directly from one transaction to a new transaction by altering the transaction mode and ID fields on the screen. IMSADF II will switch to a key selection or Data Display screen for the new transaction, depending on the information supplied by the user.

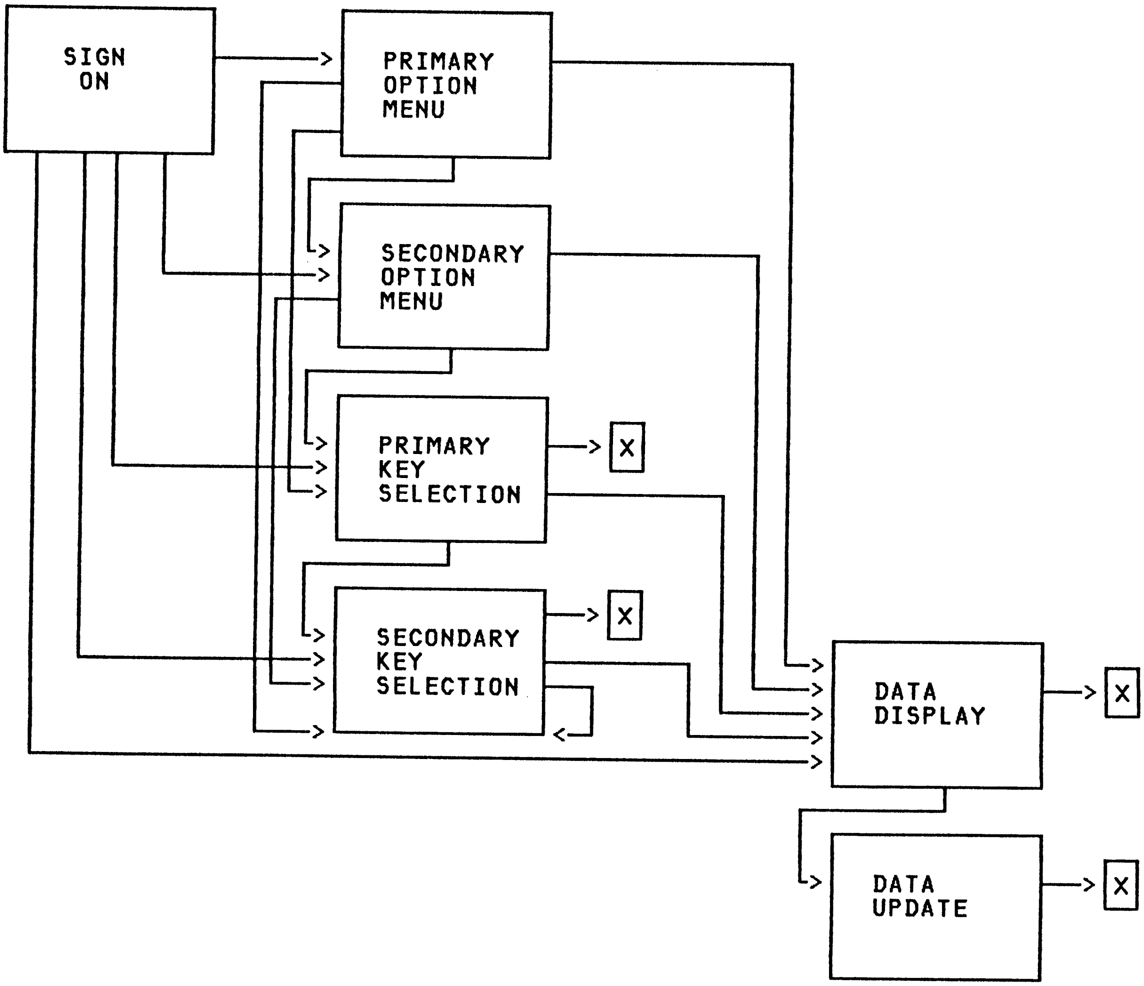


Figure 1-12. Sequence of Screens (with Variations)

Transaction switching may also be controlled by dynamic rules written when the transaction is developed. Chapter 4, "The Auditor and the Audit Data Base" explains how such rules are written.

CHAPTER 2. STATIC RULES AND THE RULES GENERATOR

Static rules are used to define:

- The transactions within an application system
- The data base and its segments
- Which segments will be used in each transaction
- What data is to appear on the display screens
- The audit specifications and interrelationships that will be required in a transaction

These rules are "static" because they are relatively stable and unchanging. The IMSADF II Rules Generator, a utility similar to a compiler, processes static rules and stores them as members in an OS partitioned data set (PDS).

STATIC RULES FOR CONVERSATIONAL APPLICATION SYSTEMS

Chapter 1, "IMSADF II Concepts and Overview" outlined the flow of menus and screens that are a standard part of conversational application systems. Figure 2-1 shows the types of rules used at each stage. Other functions (such as text editing and batch and nonconversational processing) use somewhat different rules and are discussed in later chapters.

After providing the sign-on and option menus in each IMSADF II application system, the common module retrieves the rules it requires according to the user's sign-on and subsequent selection of transactions.

An application system contains one of each of the menu rules. The menu rules are:

Primary Option Menu Rule
: Contains a list of the options (selected from A, B, C, D, F, H, I) to be displayed on the Primary Option Menu screen.

Secondary Option Menu Rule
: Contains a list of all transaction IDs in an application system. IMSADF II reconciles this list with the user's individual security profile to produce the list of transactions that will appear on the Secondary Option Menu screen.

The Primary Option Menu Rule and the Secondary Option Menu Rule must thus be generated only once for an application system. Keep in mind, though, that the Secondary Option Menu Rule must be updated whenever new transactions are added to the application system.

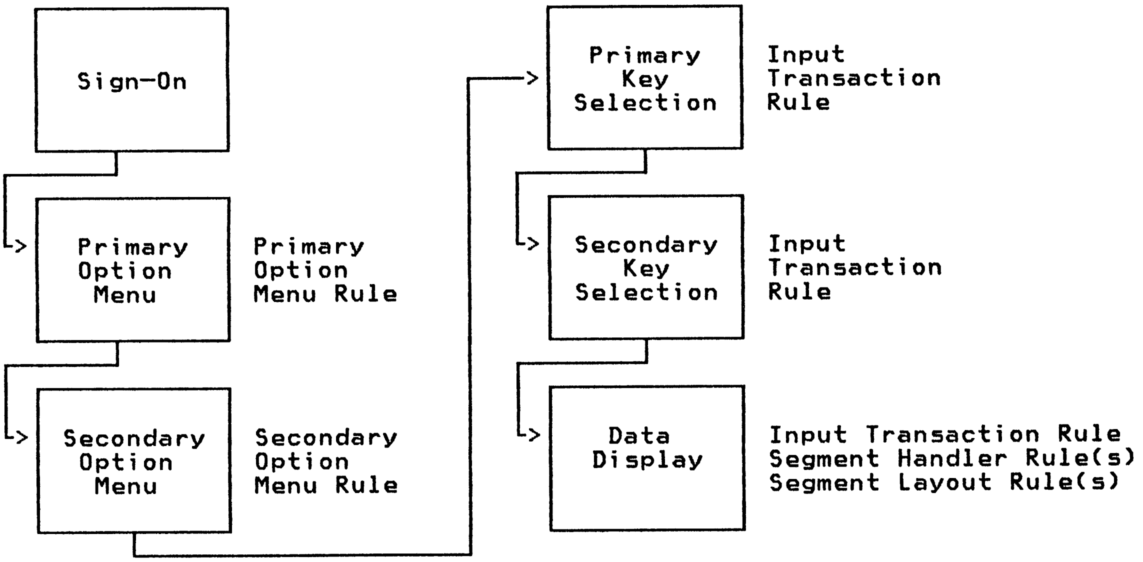


Figure 2-1. Conversational Rule Usage

Transactions are governed by the generalized application program, known as a transaction driver. One or more of the remaining static rules are built for each transaction or segment in an application system. These rules are:

Input Transaction Rule
: Defines the segments to be used in a transaction, including a small amount of information about the kind of processing to be performed against the data base.

Segment Handler Rule
: Contains the actual segment search arguments (SSAs) that IMSADF II needs to perform data base I/O using DL/I. One is produced for every data base segment to be used.

Segment Layout Rule
: Defines the fields in a segment, including their length and format, and indicates whether any validation or message sending is to be performed.

Table Handler Rule
: Builds an Assembler program containing standard static SQL calls and USER SQL calls.

Table Layout Rule
: Defines the columns in a DB2 table. It performs the same function as the Segment Layout Rule.

There are six types of source statements to be submitted to the Rules Generator. They are:

SYSTEM Defines the application system ID the user must give to obtain the Sign-On screen and sets general system parameters.

SEGMENT Defines the data base layout (similar to an IMS/VS DBD); segments are usually defined in hierarchical order. There must be a separate SEGMENT statement for every data base segment used in a transaction.

FIELD Defines the key and data fields contained in a segment and indicates how the fields are to be displayed on the screens in which they appear.

GENERATE Has several uses:

- Defines transactions, controls the screen formats, and identifies which data base segments are to be used

- Controls the generation of Segment Handler and Segment Layout Rules (using information given in the SEGMENT and FIELD statements)
- Facilitates maintenance of the list of transactions in the application system by adding new transaction IDs into the Secondary Option Menu Rule; creates a new Secondary Option Menu Rule for a new application system
- Generates the Sign-On screen and controls the Primary Option Menu screen
- Requests link edit and preload performance options

RULE Provides control information to the Rules Generator for entering Assembler language rules source.

Note: The RULE statement is not supported under the Interactive Application Development Facility (IADF).

INCLUDE Provides a copy library facility that allows basic information to be stored, retrieved, and augmented or overridden by additional statements and parameters.

Figure 2-2 shows how these Rules Generator statements can be used to produce a single transaction, PA, which allows display and update (including insertion and deletion) of the PARTROOT segment (discussed in Chapter 1, "IMSADF II Concepts and Overview").

```
//NAME     JOB ACCNT,NAME,MSGLEVEL=1
//STEP1    EXEC   ????G
//G1.SYSIN DD *
SYSTEM     SYSID=SAMP,PGROUP=ZZ,SOMTX=OR,
           SHEADING='S A M P L E   P R O B L E M'
SEGMENT    ID=PA,LENGTH=50,NAME=PARTROOT,PARENT=0
FIELD      ID=KEY,KEY=YES,LENGTH=17,NAME=PARTNUMB,
           SNAME='PART NUMBER'
FIELD      ID=DESC,LENGTH=20,POSITION=27,
           SNAME='PART DESCRIPTION'
GENERATE   OPT=CVALL,TRXID=PA,
           DBPATH=PA,TRXNAME='PART SEGMENT'
GENERATE   OPT=SGALL
GENERATE   OPT=SOM
GENERATE   OPT=CVSYS
GENERATE   OPT=STLE,PGMID=OR
```

Figure 2-2. Using Rules Generator Statements to Produce a Simple Transaction

The JCL procedure ????G executes the Rules Generator, where **????** is the installed ADFID (the default is MFC1; refer to the IMS Application Development Facility II Version 2 Release 2 Installation Guide). This procedure is supplied with IMSADF II.

The next sections describe the Rules Generator source statements shown in Figure 2-2 and give information you need to start using them. Refer to the IMS Application Development Facility II Version 2 Release 2 Application Development Reference for detailed descriptions of the various operands used in each type of statement.

THE SYSTEM STATEMENT

```
SYSTEM    SYSID=SAMP,PGROUP=ZZ,SOMTX=OR,
          SHEADING='S A M P L E   P R O B L E M'
```

The main operands of this statement are described below.

SYSID Names the application system ID, the first four characters of the program load module for this application. **Required.**

The DL/I PSB, where applicable, and the IMS/VS transaction code, both have the same name as the program load module. The first two characters of the application system ID must be unique within the installation.

PGROUP Name of project/group using this application system. This code must be unique within the installation and should be entered, together h with relevant user IDs and profiles, into the Sign-On Profile data base (see Chapter 3, "Sign-On Security"). More than one project/group can use the same application system. If PGROUP is omitted from the SYSTEM statement, it must be specified on every GENERATE statement.

SOMTX Defines the last two characters of the program load module for this application (eg. SAMPTOR). **Required.**

Note: SOMTX on the GENERATE statement (OPT=CVALL) overrides the operand on the SYSTEM statement.

SHEADING The heading that appears on the Sign-On, Key Selection, and Data Display screens.

THE SEGMENT STATEMENT

```
SEGMENT   ID=PA,LENGTH=50,NAME=PARTROOT,PARENT=0
```

All of the operands described below are **required**.

ID The two-character segment ID; must be unique within the application system.

LENGTH Segment length in bytes.

NAME Name of segment to be used in segment search arguments for DL/I calls. The same NAME value may be used for different segment definitions with different IDs. Such definitions are called aliases and are different views of the same data base segment.

PARENT The two-character ID of the parent segment in the data base. Root segments should have PARENT=0. DB2 tables and VSAM files should also have PARENT=0.

THE FIELD STATEMENT

```
FIELD     ID=KEY,KEY=YES,LENGTH=17,NAME=PARTNUMB,
          SNAME='PART NUMBER'
FIELD     ID=DESC,LENGTH=20,POSITION=27,
          SNAME='PART DESCRIPTION'
```

All FIELD statements must have an ID and a LENGTH.

ID Field ID; up to four characters; must be unique within the segment. **Required.**

KEY Indicates whether or not the field is a key field. The default is NO.

BYTES or LENGTH Length of stored field in bytes. **Required.**

NAME Name of field to be used in segment search arguments for DL/I calls.

TYPE Defines data type (see Figure 2-3). The default is ALPHANUM. Note that type NUMeric allows numeric characters 0 to 9 only (not signs or decimal points) and is used rarely. Data used for arithmetic operations by the Auditor (see Chapter 4, "The Auditor and the Audit Data Base") must be DEC (zoned decimal), PD (packed decimal), BIN (binary) or NUMeric.

START or POSITION Position of field in the segment. The default is the byte immediately following the field defined in the preceding FIELD statement; if this is the first FIELD statement in the segment, the default is position 1.

SNAME The heading that appears with the field on the key selection and Data Display screens.

Notation	Abbreviation	Meaning	Used for Arithmetic Operations
ALPHA	C	Alphabetic characters and blank	NO
ALPHANUM	C	Alphanumeric characters	NO
BIN	I	Binary number	YES[1]
BIT	B	Bit data	NO
DATE	DA or D	Date	NO
DBCS	DB	Double Byte Character Set	NO
DEC	DE or Z	Zoned decimal	YES[1]
FLOAT	F	Floating point	YES
HEX	H or X	Hexadecimal presentation	NO
MIXED	M	Mixed EBCDIC and DBCS	NO

Figure 2-3 (Part 1 of 2). Data Types

Notation	Abbreviation	Meaning	Used for Arithmetic Operations
NUM	N	Numeric digits	YES[2]
PD	P	Packed decimal	YES[1]
UDEC	UD or UZ	Unsigned zoned decimal	YES[2]
UPD	UP	Unsigned packed decimal	YES[2]
VARCHAR	V	Variable length character	NO

Figure 2-3 (Part 2 of 2). Data Types

Key Fields

Key fields are identified to IMSADF II by marking them KEY=YES in the FIELD statement.

Processing with IMSADF II is simplest if every data base segment has a unique key sequence field. This should be considered when designing data bases for new applications. If existing data bases are involved where non-keyed segments or segments with non-unique keys exist, see "Processing Non-Keyed Segments" in the IMS Application Development Facility II Version 2 Release 2 Application Development Reference for a discussion of the supported functions and processing capabilities for those situations.

Key fields will be identified in the IMS/VS DBD as follows:

```
  FIELD NAME=(xxxxxxxx,SEQ,U),START=..,BYTES=..
```

The key field may be defined to the IMSADF II Rules Generator in the same way. If this is done, the KEY=YES operand is not required because it is implied by the FIELD statement SEQ operand. The field must, however, be given an ID. Here are two equivalent forms of the Rules Generator statements for defining key fields:

```
   FIELD  NAME=(PARTNUM,SEQ,U),START=5,BYTES=25,ID=PTNO
   FIELD  NAME=PARTNUM,KEY=YES,POS=5,LENGTH=25,ID=PTNO
```

It is assumed here that the key field is alphanumeric and has a displayed length (SLENGTH) equal to the stored length of 25 bytes.

The Rules Generator requires that every declared data base segment have a key field identified in one of the two ways just shown. If the field so identified to IMSADF II is a search field (and hence is defined in the DBD without the SEQ operand value) rather than a unique key sequence field, IMSADF II will still handle it, as long as:

- The segment is the lowest retrieved in its hierarchical path
- The search field contains unique values identifying each segment occurrence

If a search field is being used, KASCEND=NO (key not ascending) must be included in the Rules Generator SEGMENT statement.

If the search fields are not unique, it will not be possible to retrieve more than the first segment occurrence with a particular search field value through the standard key selection process. Such retrievals can

[1] Displayed right justified, with leading minus sign when negative and with leading zeros suppressed.

[2] Will not maintain sign if negative

be handled using the Auditor, as explained in Chapter 6, "Complex Transactions."

It is possible to divide the key field into contiguous subfields. Then the key selection and Data Display screens will show the subfields instead of a single long key field for that segment. To achieve this, define each subfield separately to the Rules Generator, each with KEY=YES. (Do not code the overall key field.) At the same time, the name by which the key field is defined in the DBD must be coded against the SEGMENT statement, not against the fields. For example:

```
SEGMENT   ID=PT,NAME=PARTROOT,LENGTH=100,PARENT=0,
          KEYNAME=PARTNUM
FIELD     ID=PTTP,KEY=YES,POS=5,LENGTH=10
FIELD     ID=PTAS,KEY=YES,LENGTH=15
```

This definition will work against the same DBD as the previous example but will format the keys differently.

Decimal Fields

When a zoned (DEC) or packed (PD) decimal field is displayed on a screen, allowance is made for a sign and a decimal point. Therefore, the default screen length (SLENGTH) for a field of length n is n+2 (TYPE=DEC) or 2n+1 (TYPE=PD), even if there are no decimal places. If a shorter or longer SLENGTH is specified, the value will be right justified and leading zeros suppressed, with a floating minus sign for negative numbers. If a shorter screen length than the default is defined and a data value that is too long is encountered on the data base during execution, asterisks will appear in the displayed value.

DEC and PD fields can have decimal points, defined on the FIELD statement using the following operands:

DECIMAL The number of decimal places. The decimal point will be displayed but not stored. Values entered by the user will automatically be aligned to the decimal point position.

SDECIMAL The number of decimal places displayed; can be different from the number assumed to be present by the implied decimal point position.

If, for example, a field is defined as:

```
FIELD   ID=FFFF,LENGTH=6,TYPE=DEC,DEC=2
```

it will be displayed on the screen as:

```
__________3_._1_0
 (8 positions)
```

The user can amend the value by entering data in front of the old value (without pressing EOF):

```
_2_._4___3_._1_0
 (8 positions)
```

Result:

```
__________2_._4_0
 (8 positions)
```

If a value that is too large to be stored is entered, an error message appears. This can happen if the user writes a digit in the sign position:

```
_1_2_3_4_5_._6_7
 (8 positions)
```

Date Fields

Data of TYPE=DATE is stored as "YYMMDD" on the data base and in the SPA. Any manipulation or comparison of a TYPE=DATE field should refer to it in this format, except when assigning a literal value in the audit rules, when the form displayed should be used.

The format in which IMSADF II displays a TYPE=DATE field will depend on the option selected by the installer of the product. The DATEFMT operand of the DEFADF macro statement (refer to the IMS Application Development Facility II Version 2 Release 2 Installation Guide) can have one of five possible values:

DATEFMT=S	International standard - YY-MM-DD
DATEFMT=B	International standard with blank separator - YY MM DD
DATEFMT=U	Former U.S. standard - MM/DD/YY
DATEFMT=E	Former European standard with dot separator - DD.MM.YY
DATEFMT=O	Old world standard - DD/MM/YY

When the user enters or amends a date field, it will be validated for a correct month number (1-12), a correct day number, depending on the month and including a check for leap year, and a numeric year number.

Controlling Display Screen Contents

Unless screen image is used (see Chapter 6, "Complex Transactions"), the following operands are commonly used in the FIELD statement to produce the Data Display screen:

SNAME Name to appear against the field on the Data Display screen. For key fields, this name will also appear on the Primary Key Selection screen. SNAME is also used for headings of Secondary Key Selection Screen if SKLEFT and SKRIGHT have not been specified.

MODE Sets the attributes for the field. These can be:

4 - Modifiable (transaction modes 1-6)
5 - Modifiable (transaction modes 1-5)
6 - Nonmodifiable
7 - Modifiable but not displayed (e.g., for lockwords)

The default for nonkey fields is modifiable (MODE=5). Key fields are, by default, nonmodifiable (MODE=6). If a key field is defined as MODE=5, the user can alter the key value displayed; this has the same effect as altering the corresponding value within the concatenated key field (which appears near the top of the screen): a new transaction starts and retrieves the segment with the new key.

DISPLAY Controls the contents of the Data Display screen. The fields and segments defined may appear in several different transactions and a particular field may be required in some and not in others. A simple default exists which ties in with the concept of the target segment; that is, the lowest level segment in each hierarchical path retrieved by a transaction. By default, the Data Display screen for a transaction will include all the fields of the target segments and the key fields of all the other segments in the hierarchical paths down to the target segments.

The default values for MODE and DISPLAY operands can be altered for all the fields in a segment by coding them on the SEGMENT statement. For example, if MODE=5 and DISPLAY=YES are included on a SEGMENT statement, all the fields (including keys) will be displayed and modifiable on display screens for all transactions in which the segment is accessed. Individual fields can still bear different values for these operands: the values on the FIELD statements can override the SEGMENT values.

An example should clarify the use of DISPLAY. We will define two transactions against the data base shown in Figure 2-4.

AKEY KEY=YES	AA11 DISP=YES	AA22 DISP=NO	AA33	AA
BKEY KEY=YES	BB11 DISP=YES	BB22 DISP=NO	BB33	BB

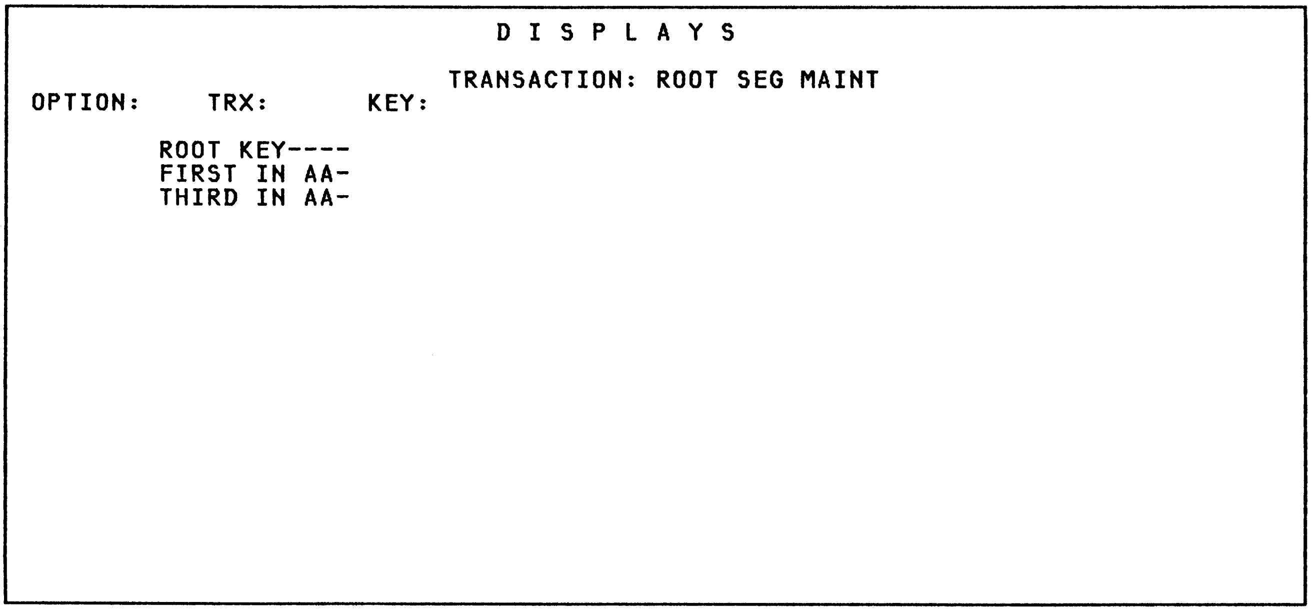
Figure 2-4. Data Base to Illustrate Use of DISPLAY Operand

The Rules Generator statements are as follows:

```
SYSTEM     SYSID=EXDI,PGROUP=PG,SOMTX=TT,SHEADING='D I S P L A Y S'
SEGMENT    ID=AA,LENGTH=40,NAME=AAROOT,PARENT=0
FIELD      ID=AKEY,KEY=YES,NAME=AASEGKEY,LENGTH=10,SNAME='ROOT KEY'
FIELD      ID=AA11,LENGTH=10,DISP=YES,SNAME='FIRST IN AA'
FIELD      ID=AA22,LENGTH=10,DISP=NO,SNAME='SECOND IN AA'
FIELD      ID=AA33,LENGTH=10,SNAME='THIRD IN AA'
SEGMENT    ID=BB,LENGTH=40,NAME=BBDEPSEG,PARENT=AA
FIELD      ID=BKEY,KEY=YES,NAME=BBSEGKEY,LENGTH=10,
           SNAME='DEPENDENT KEY'
FIELD      ID=BB11,LENGTH=10,DISP=YES,SNAME='FIRST IN BB'
FIELD      ID=BB22,LENGTH=10,DISP=NO,SNAME='SECOND IN BB'
FIELD      ID=BB33,LENGTH=10,SNAME='THIRD IN BB'
*NOW GENERATE THE SEGMENT RULES
GENERATE   SEG=(AA,BB),OPTIONS=SGALL
*NOW GENERATE THE TRANSACTION RULES AND SCREENS
GENERATE   TRXID=RO,TRXNAME='ROOT SEG MAINT',DBPATH=AA,
           OPTION=CVALL
GENERATE   TRXID=DE,TRXNAME='DEP SEG MAINT',DBPATH=BB,
           OPTION=CVALL
*NOW ADD THE NEW TRANSACTION IDs TO THE SECONDARY OPTION MENU RULE
GENERATE   OPT=SOM
```

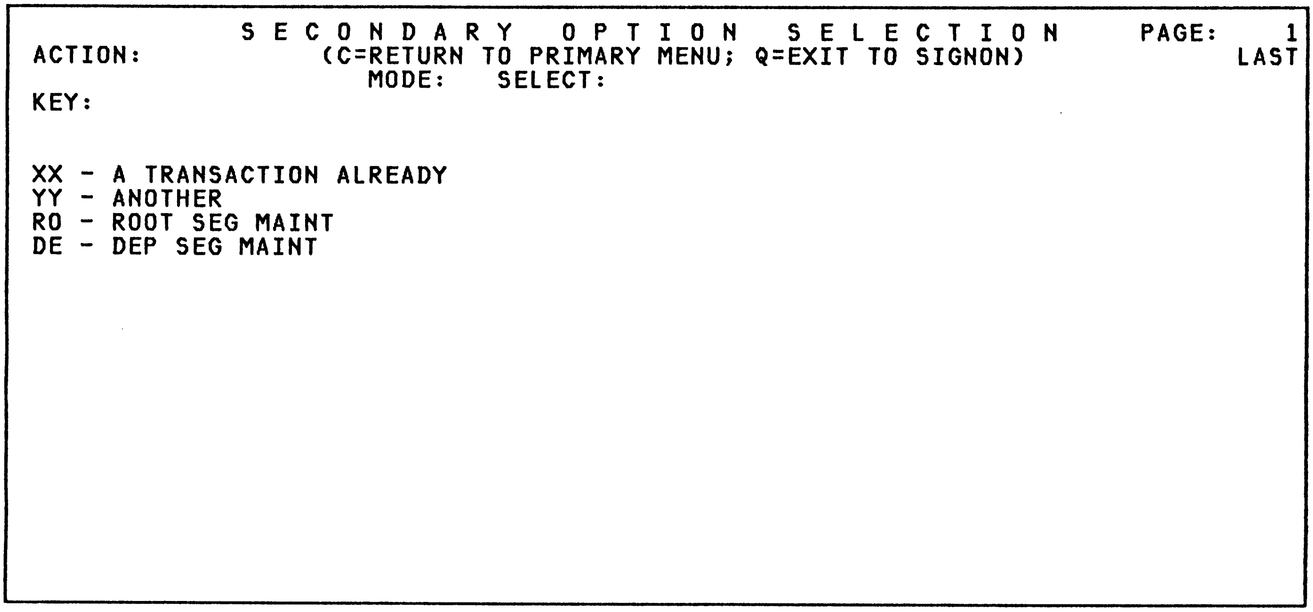
The resulting Data Display screens (requested by the CVALL option) are shown in Figure 2-5 and Figure 2-7, while the Primary Key Selection screens (also requested by the CVALL option) are in Figure 2-8 and Figure 2-9.

The Secondary Option Menu screen is formatted dynamically by the system depending on the user's security profile and the Secondary Option Menu Rule. The Secondary Option Menu screen will list all the transactions in an application system available to that user, along with a brief description obtained from the TRXNAME operand of the GENERATE statement. GENERATE OPT=SOM adds the two new transactions to the existing list. Assuming that the user is authorized to use them, the Secondary Option Menu screen will look like the one in Figure 2-6, where XX and YY are transactions already in the system.



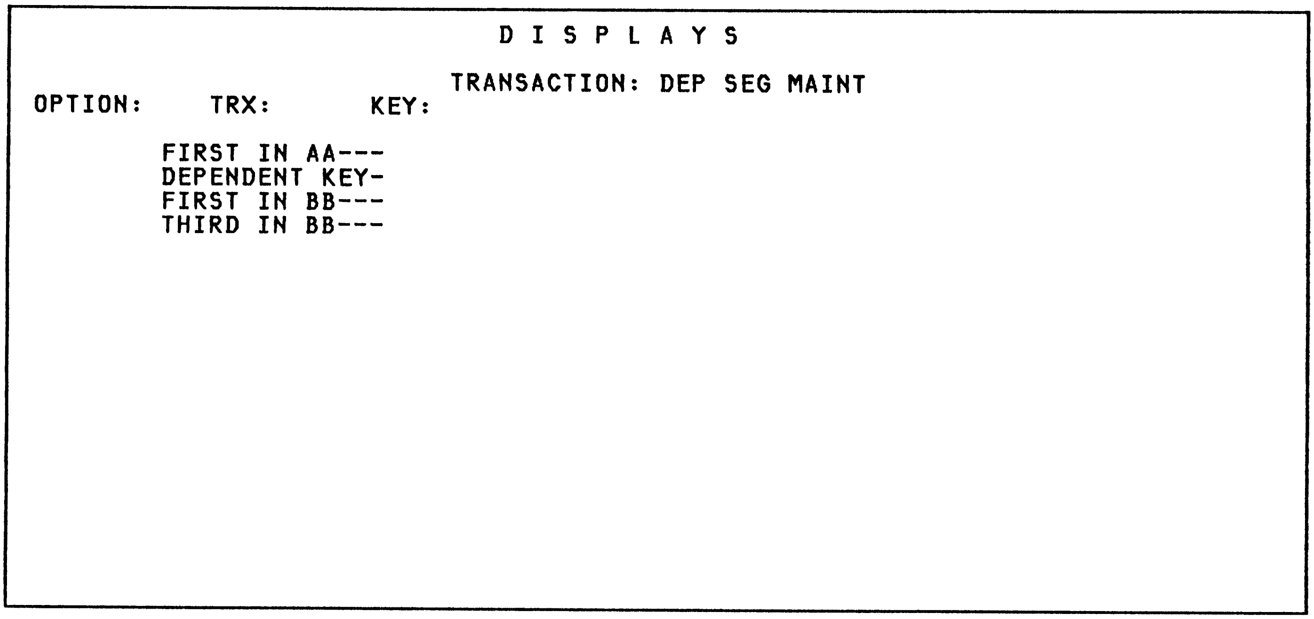
```
                               D I S P L A Y S
                           TRANSACTION: ROOT SEG MAINT
 OPTION:    TRX:      KEY:

        ROOT KEY----
        FIRST IN AA-
        THIRD IN AA-
```

Figure 2-5. Data Display Screen for Root Maintenance Transaction



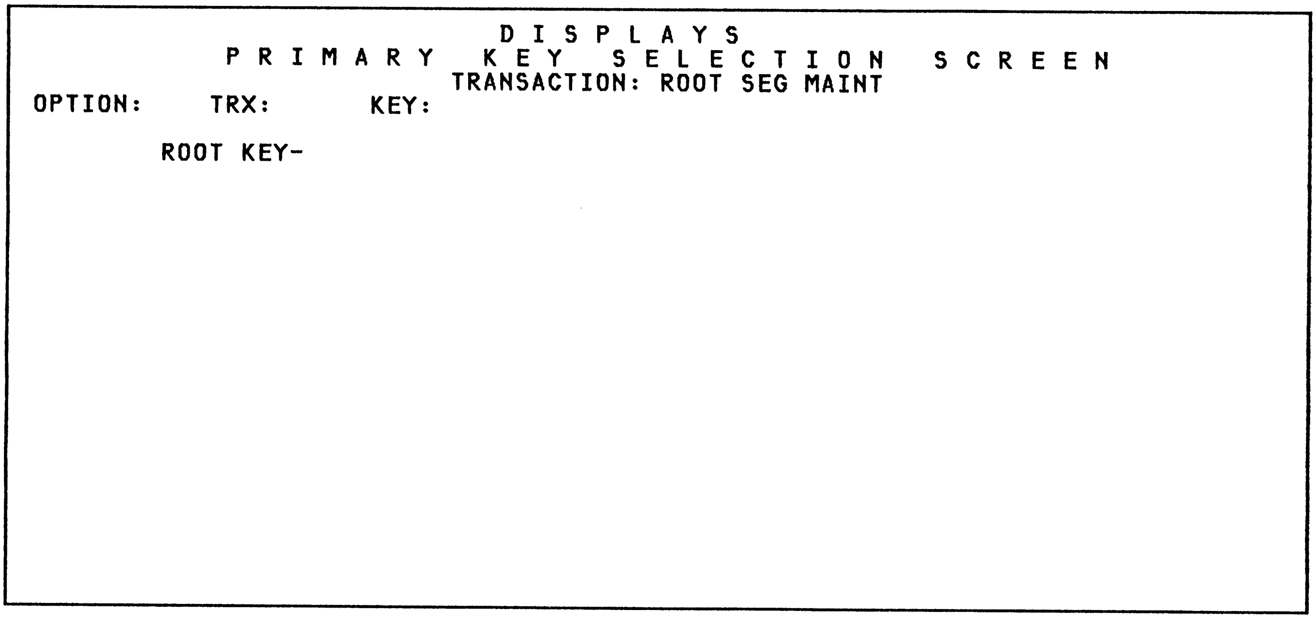
```
                S E C O N D A R Y   O P T I O N   S E L E C T I O N    PAGE:   1
 ACTION:            (C=RETURN TO PRIMARY MENU; Q=EXIT TO SIGNON)             LAST
                       MODE:   SELECT:
 KEY:


 XX - A TRANSACTION ALREADY
 YY - ANOTHER
 RO - ROOT SEG MAINT
 DE - DEP SEG MAINT
```

Figure 2-6. The Updated Secondary Option Menu



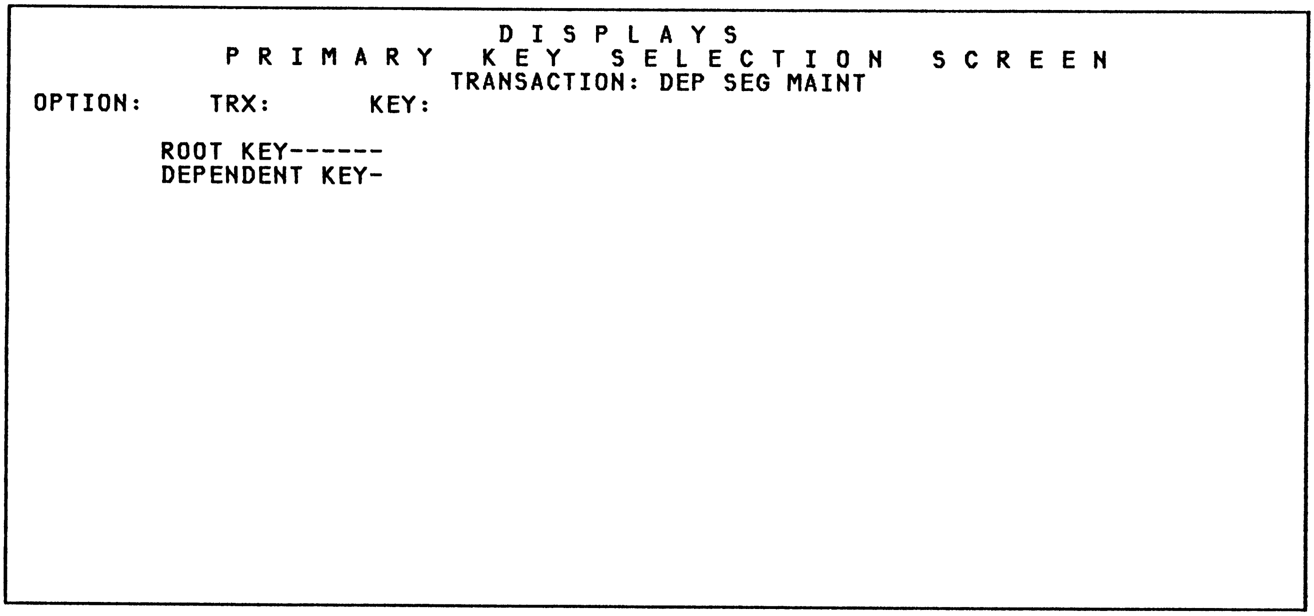
```
                              D I S P L A Y S
                          TRANSACTION: DEP SEG MAINT
OPTION:     TRX:       KEY:

        FIRST IN AA---
        DEPENDENT KEY-
        FIRST IN BB---
        THIRD IN BB---
```

Figure 2-7. Data Display Screen for Dependent Segment Maintenance Transaction



```
                              D I S P L A Y S
            P R I M A R Y   K E Y   S E L E C T I O N   S C R E E N
                          TRANSACTION: ROOT SEG MAINT
OPTION:     TRX:       KEY:

        ROOT KEY-
```

Figure 2-8. Primary Key Selection Screen for Root Maintenance Transaction



```
                                D I S P L A Y S
              P R I M A R Y   K E Y   S E L E C T I O N   S C R E E N
                             TRANSACTION: DEP SEG MAINT
OPTION:    TRX:      KEY:

        ROOT KEY------
        DEPENDENT KEY-
```

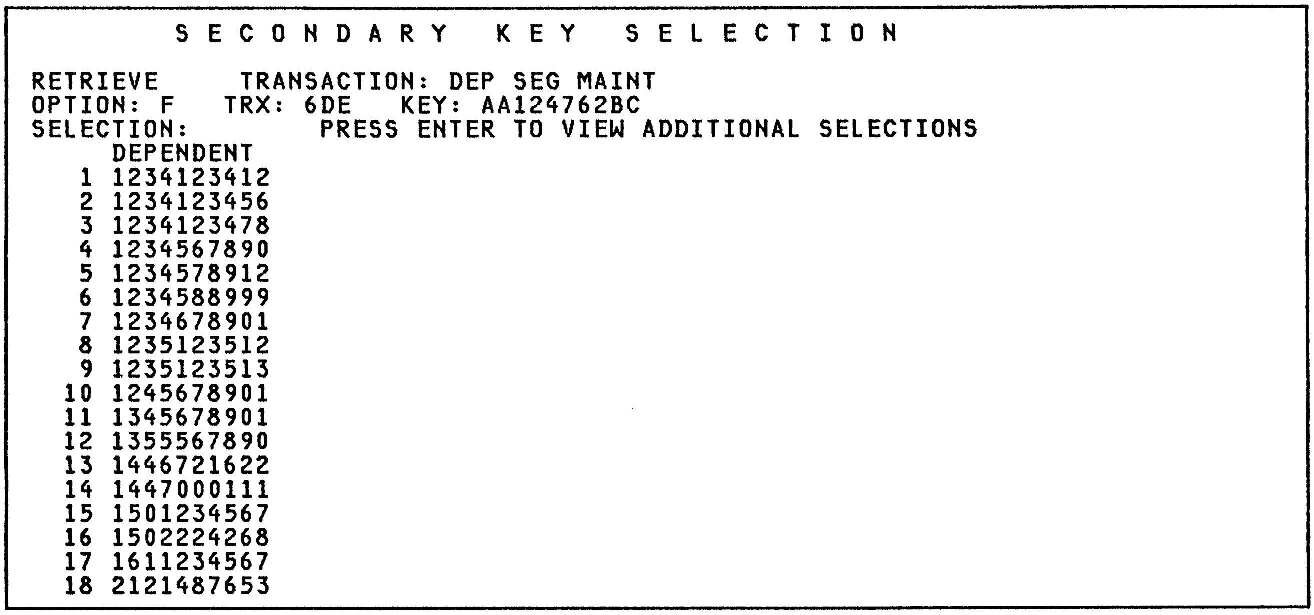
Figure 2-9. Primary Key Selection Screen for Dependent Segment Maintenance Transaction

Secondary Key Selection

Secondary key selection, a data base browsing capability, can be provided as part of every IMSADF II transaction.

Browsing is available at every segment level and in every hierarchical path accessed by a transaction. The screens themselves are formatted dynamically by the key selection module but are controlled by certain Rules Generator operands.

By default, IMSADF II will perform secondary key selection on dependent segments but not on root segments. The format of the screen for browsing through BB segments, following the example of the previous section, is shown in Figure 2-10. Each line in the example shows the key of one segment occurrence under a particular root key. The column heading ("DEPENDENT") is derived from the SNAME of the key field but is shortened to the length of the key field itself.



```
          S E C O N D A R Y   K E Y   S E L E C T I O N

 RETRIEVE     TRANSACTION: DEP SEG MAINT
 OPTION: F   TRX: 6DE   KEY: AA124762BC
 SELECTION:         PRESS ENTER TO VIEW ADDITIONAL SELECTIONS
      DEPENDENT
    1 1234123412
    2 1234123456
    3 1234123478
    4 1234567890
    5 1234578912
    6 1234588999
    7 1234678901
    8 1235123512
    9 1235123513
   10 1245678901
   11 1345678901
   12 1355567890
   13 1446721622
   14 1447000111
   15 1501234567
   16 1502224268
   17 1611234567
   18 2121487653
```

Figure 2-10. Default Secondary Key Selection Screen

The browsing function reads and displays the first 18 segment occurrences under the given root key (already entered by the user) and invites the user to press ENTER if there are more occurrences. The maximum number of segment occurrences displayed will depend on the model and type of display used. The screen format can be varied by means of the following two SEGMENT statement operands.

SKSEGS Number of occurrences to be displayed on the Secondary Key Selection screen. The default is 18 unless an extra line of heading is requested, which reduces the default to 17. If a value of zero is coded, no secondary key selection will be performed for this segment type. The default value for root segments is zero. Secondary key selection for root segments, when requested via a nonzero value of SKSEGS, supports partial or generic key retrieval.

SKLEFT and SKRIGHT Fuller headings can be requested. One or two lines of column headings are allowed, each up to 72 bytes in length. The first character of the heading starts in column 9 of the Secondary Key Selection screen. Define the first 36 bytes of the heading as the value of SKLEFT and the second 36 bytes as the value of SKRIGHT. For a second line of heading, code the operands again.

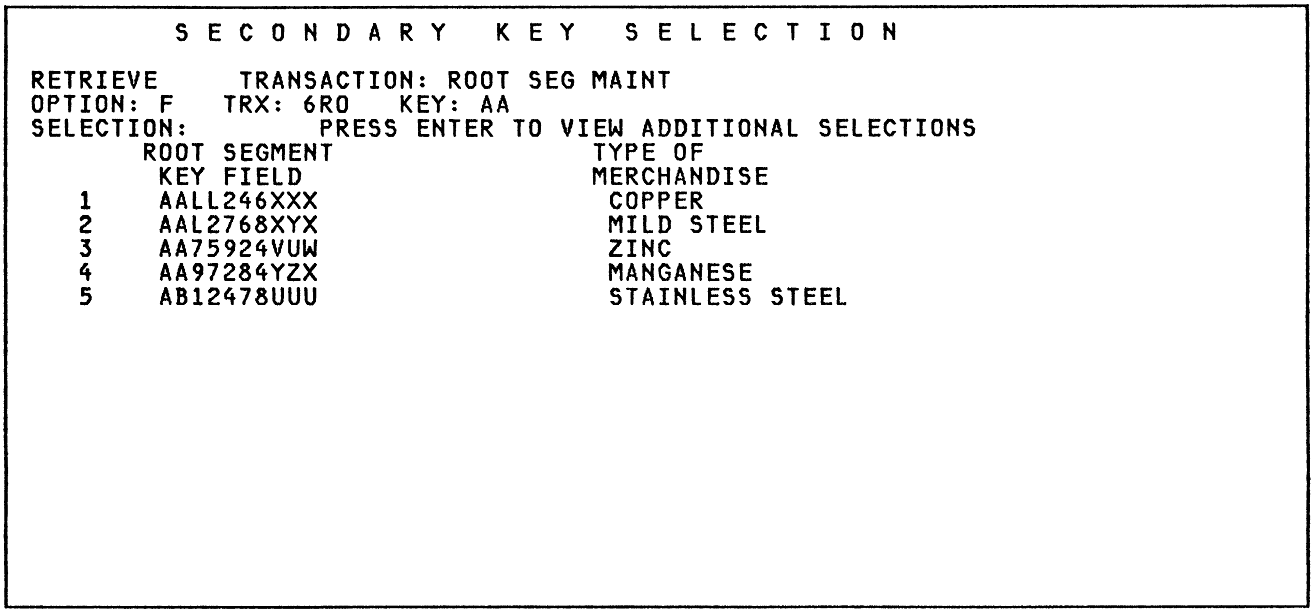
It is possible to request additional fields from the segment to be displayed on the Secondary Key Selection screen beside the key on each line. This is done on the FIELD statement by the RELATED=YES operand.

RELATED COL Defines the starting column number of the displayed field on each line. Column 1 is aligned with the first character of the heading. By default, the key field starts in column 1 and each subsequent field starts 2 bytes after the end of the previous one.

If the definition of the AA root segment in the example to request browsing and define headings is amended, the following modifications must be made to the Rules Generator statements:

```
SEGMENT   ID=AA,LENGTH=40,NAME=AAROOT,PARENT=0,SKSEGS=5,
          SKLEFT='ROOT SEGMENT',SKRIGHT='TYPE OF',
          SKLEFT=' KEY FIELD',SKRIGHT='MERCHANDISE'
FIELD     ID=AKEY,KEY=YES,NAME=AASEGKEY,
          LENGTH=10,SNAME='ROOT KEY',COL=2
FIELD     ID=AA11,LENGTH=10,DISP=YES,SNAME='FIRST IN AA'
FIELD     ID=AA22,LENGTH=10,DISP=NO,SNAME='SECOND IN AA',
          RELATED=YES,COL=38
FIELD     ID=AA33,LENGTH=10,SNAME='THIRD IN AA'
```

Figure 2-11 shows the Secondary Key Selection screen that results.



```
          S E C O N D A R Y   K E Y   S E L E C T I O N

RETRIEVE     TRANSACTION: ROOT SEG MAINT
OPTION: F   TRX: 6RO   KEY: AA
SELECTION:        PRESS ENTER TO VIEW ADDITIONAL SELECTIONS
       ROOT SEGMENT                    TYPE OF
        KEY FIELD                      MERCHANDISE
   1    AALL246XXX                      COPPER
   2    AAL2768XYX                      MILD STEEL
   3    AA75924VUW                      ZINC
   4    AA97284YZX                      MANGANESE
   5    AB12478UUU                      STAINLESS STEEL
```

Figure 2-11. Tailored Secondary Key Selection Screen

THE GENERATE STATEMENT

```
GENERATE  OPT=CVALL,TRXID=PA,
          DBPATH=PA,TRXNAME='PART SEGMENT'
GENERATE  OPT=SGALL
GENERATE  OPT=SOM
GENERATE  OPT=CVSYS
GENERATE  OPT=STLE,PGMID=OR
```

This example includes five types of GENERATE statements. The OPTIONS operand (OPT) determines which kind it is.

1. The first GENERATE statement (OPT=CVALL) generates a an IMSADF II transaction. The DBPATH operand gives information about key selection; in this case the PA segment is named. From the definition of PA, the Rules Generator can produce the formatting information required for the Primary and Secondary Key Selection screens and the Data Display screen.

2. The second GENERATE statement (OPT=SGALL) generates all the segment rules needed for all segments defined in this example before the GENERATE statement is encountered.

3. The third GENERATE statement (OPT=SOM) adds the new transaction ID (and the TRXNAME value) to the list of existing transaction IDs held in the Secondary Option Menu Rule for this application system. If

this is the first time this application system has been defined, a new rule will be created. Otherwise, the existing rule will be amended or extended with the TRXID and TRXNAME values found in this example.

4. The fourth GENERATE statement (OPT=CVSYS) generates the Primary Option Menu Rule and the Sign-On screen.

5. The last GENERATE statement (OPT=STLE) requests the link edit of an application 'mini-driver' program. The PGMID is the same as the cluster code (SOMTX) parameter. This GENERATE statement is required only once for each cluster code.

IMSADF II transactions are defined by the GENERATE statement with OPT=CVALL. The main operands of this statement are described below.

TRXID Names the two-character transaction ID which is to appear on the list on the Secondary Option Menu screen. The user invokes the transaction by entering this code. **Required.**

TRXNAME Sets the descriptive name of the transaction that will appear on the Secondary Option Menu screen, the Key Selection screens, and (unless screen image is used) the Data Display screen.

DBPATH Defines the target segments of the transaction. These are the segments for which the user will be prompted to enter key information through key selection and which will be retrieved and updated according to the transaction mode selected by the user at the terminal.

TSEGS Names working storage areas (called pseudo segments) and data base segments that are to be retrieved and updated under control of dynamic rules.

DEVNAME Indicates the name of the terminal type to be used as assigned during IMS/VS system definition. Possible values are 2 or Ann where nn is a one- or two-digit number.

DEVTYPE Indicates the characteristics of the terminal as follows:

2	3270 display with a 24 x 80 screen
3	3278 model display with a 32 x 80 screen
4	3278 model display with a 43 x 80 screen
5	3278 model display with a 24 x 132 screen
6	3279 model 2B color display with a 24 x 80 screen
7	3279 model 3B color display with a 32 x 80 screen
8	3290 Information Display panel with a 62 X 160 screen
9	5555 Multi-station Display with a 24 x 80 screen

Operands required to use screen image definitions and color or to change the default processing against data base segments are described in Chapter 6, "Complex Transactions."

The rules associated with segments (the Segment Layout and Segment Handler rules) will be generated when the Rules Generator encounters the following statement:

```
  GENERATE  OPT=SGALL
```

It should be placed after all SEGMENT statements in the input to the Rules Generator.

Whenever new transactions (new TRXIDs) are created or descriptive names (TRXNAMEs) are changed, the Secondary Option Menu Rule - which controls the contents of the transaction list that appears on the Secondary Option Menu screen - must be updated. The Rules Generator will do this when it encounters the following statement:

```
  GENERATE  OPT=SOM
```

It should be placed after all transaction definitions.

The last two GENERATE statements are:

```
GENERATE  OPT=CVSYS
GENERATE  OPT=STLE,PGMID=OR
```

The main operands of these statements are described below.

PGMID Defines the cluster code and hence the last two characters of the IMS/VS transaction code. One of these GENERATE statements must match the SOMTX operand value on the SYSTEM statement. Any IMSADF II transaction that uses a different cluster code must have SOMTX on the GENERATE (with TRXID) statement, thus overriding the operand on the SYSTEM statement. There must be a separate GENERATE PGMID statement for each different SOMTX operand value. **Required.**

POMENU On the GENERATE OPT=CVSYS statement, selects from options (A, B, C, D, F, H, I) to appear on the Primary Option Menu screen. The default is all of them.

PSEUDO SEGMENTS

Sometimes it is necessary to define working storage in a transaction for calculations or other data manipulation. Sometimes the fields in working storage will be displayed and possibly updated by the user on the Data Display screen. IMSADF II provides pseudo segments for this purpose. They are defined to the Rules Generator similar to data base segments but without key fields, without parents, and without NAME operands. On the SEGMENT statement, TYPE=PS (pseudo segment) must be specified. By default, the fields are displayed with MODE 5 but this can be changed just as for data base segments. A pseudo segment which resides in the conversational communications area is called a COMM segment and is specified with TYPE=COMM on the SEGMENT statement.

In order to use a pseudo segment in a transaction it must be named in the TSEGS operand of the GENERATE statement for that transaction. Here is an example:

```
SEGMENT   ID=CC,TYPE=PS
FIELD     ID=CLOR,TYPE=DEC,LENGTH=7,SNAME='CLOSE ORDER'
GENERATE  TRXID=UV,TRXNAME='PROCESS ORDERS',DBPATH=IV,TSEGS=CC,
          OPTIONS=CVALL
```

SUMMARY OF SYNTAX CONVENTIONS

- Start in any column (1-71) and use columns 1 to 71.
- Leave one or more spaces between control statement keywords and operands.
- Separate operands by commas with no intervening blanks.
- Separate comments from statements by one or more blank lines. An asterisk in column 1 marks a comment line.
- Mark continuations by a comma/blank combination. The next line can start in any column (1-71).
- Do not continue multi-valued operands (using parentheses) over two lines. Instead, close the parentheses and repeat the operand on the next line: TSEGS=AA,TSEGS=BB is the same as TSEGS=(AA,BB).

ABBREVIATIONS

All keywords can be abbreviated to the minimum number of initial letters that makes them unique. The following abbreviations are common.

- On the FIELD statement:

LENGTH	- LEN	SLENGTH	- SL
TYPE	- TY	POSITION	- POS
DECIMAL	- DEC	SDECIMAL	- SDEC
SNAME	- SN	DISPLAY	- DISP

```
        RELATED  - REL
```

- On the GENERATE statement:

```
        OPTIONS  - OPT
        SEGMENTS - SEG
        TSEGS    - TSEG
        DBPATH   - DBP
```

MANAGING APPLICATION DEVELOPMENT AND MAINTENANCE

It is important to organize source statements for an application system so that it is easy to recreate them when it is time to move from test to production. When a large application system is being developed by several programmers, the allocation of transaction and segment IDs and aliases must be controlled, and all programmers must be working with common definitions.

Therefore a set of **master rules** should be prepared to define the layout of all data base segments to be used. The master rules can then be used to develop a basic set of data base maintenance transactions, one transaction for every segment type. This will provide the bulk of the input for more complex transaction definitions. Moreover, transactions so developed are available as soon as the data base design is complete and can be used for end user demonstrations, for loading test data, for data base maintenance in production, and even for online data entry if the volumes of data are suitable. Such data base maintenance transactions often constitute a substantial part of the final application system and can be prepared in an extremely short time using IMSADF II.

The master rules may be put into a copy library and copied into the input stream (using the INCLUDE statement) for use in creating other transactions. This will ensure that everyone will use a common set of basic definitions where multiple transactions use common segments.

The master rules operands may be expanded or overridden by statements following the included member. These overrides change or add parameters necessary to generate the desired transactions. For example, the INCLUDE library could contain the following two members:

- Member SAMPSYS:

```
    SYSTEM     SYSID=SAMP,PGROUP=ZZ,SOMTX=OR,
               SHEADING='S A M P L E   P R O B L E M'
    GENERATE   OPT=CVSYS
    GENERATE   OPT=STLE,PGMID=OR
```

- Member SAMPPA:

```
    SEGMENT    ID=PA,LENGTH=50,NAME=PARTROOT,PARENT=0
    FIELD      ID=KEY,KEY=YES,LENGTH=17,NAME=PARTNUMB,
               SNAME='PART NUMBER'
    FIELD      ID=DESC,LENGTH=20,POS=27,
               SNAME='PART DESCRIPTION'
```

The following input to the Rules Generator JCL procedure MFC1G would incorporate these statements into a new transaction, in which the DESC field is to appear (REL=YES) on the Secondary Key Selection screen:

```
  INCLUDE    SAMPSYS
  INCLUDE    SAMPPA
  FIELD      ID=DESC,REL=YES
  GENERATE   OPT=CVALL,TRXID=PA,
             DBPATH=PA,TRXNAME='PART SEGMENT'
  GENERATE   OPT=SGALL
  GENERATE   OPT=SOM
```

The library containing the included members must be named in the MFC1G JCL procedure with DDNAME equal to ADFLIB.

CHAPTER 3. SIGN-ON SECURITY

The IMSADF II data bases can be in one of two formats. They can be either DL/I hierarchical data bases or DB2 relational data bases. This chapter describes the DL/I form of the Sign-on Profile Data Base. Information about the maintenance of the DB2 data bases is found in Chapter 4, "Dynamic Rules Data Bases" of the IMS Application Development Facility II Version 2 Release 2 Application Development Reference.

The Sign-On Profile Data Base is one of three IMSADF II data bases. The rules stored in this data base are used to verify user ID, project/group code, and application system ID when a user signs on to a conversational application system. This data base also contains the user's security profile, which is a list of transaction IDs (representing the transactions a user is allowed to access) and the mode (1-6) allowed for each.

This data base is required in conversational application systems. Most transactions using dynamic rules also require information stored in the Audit and Message Data Bases (see Chapter 4, "The Auditor and the Audit Data Base").

Entering rules into the Sign-On Profile Data Base should be the responsibility of the person in charge of security, normally the data base administrator.

CONTROLLING SECURITY PROFILES ONLINE

When signing on, the user enters a user ID and a two-character project/group code. Figure 3-1 shows how users, project/groups, and application systems interrelate.

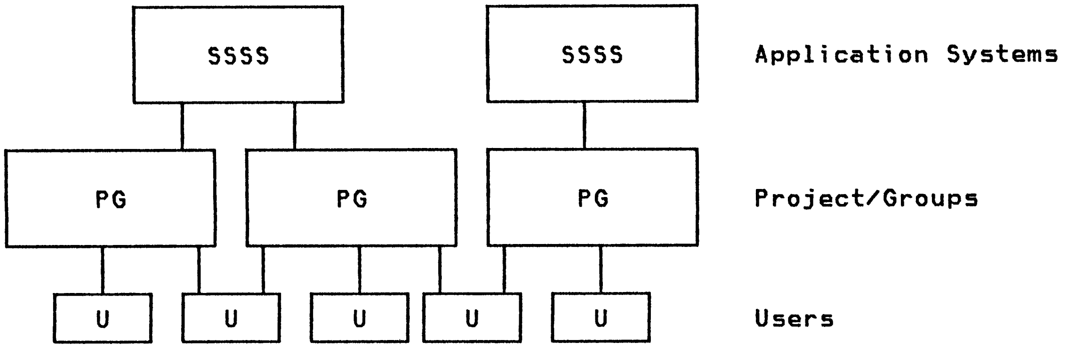


Figure 3-1. Relationship Among Application Systems, Project/Groups, and Users

Each project/group can use only one application system, but an application system can be used by many project/groups. In simple cases, there will be a one-to-one correspondence between application system IDs and project/group codes. Each must be unique in the installation. In fact, the first two characters of the application system ID must be unique. It is permissible for the project/group code to be equal to the first two characters of the application system ID, although this may not be in the best interests of security.

User IDs are different. The same user can be in several project/groups using the same or different application systems; he can have a different security profile in each project/group.

Figure 3-2 shows the structure of the Sign-On Profile Data Base.

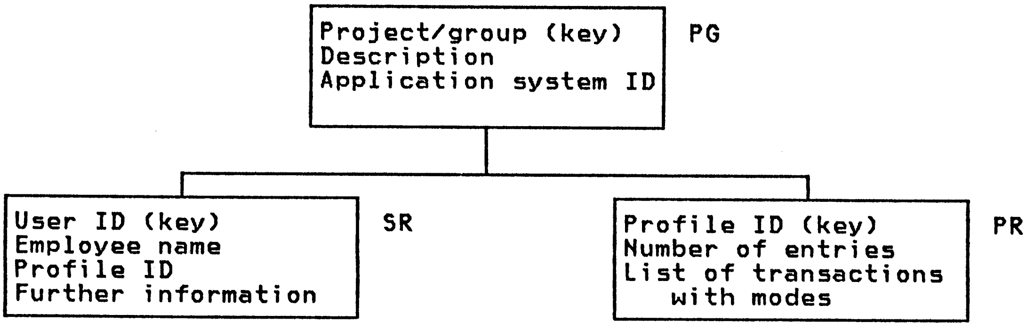


Figure 3-2. Sign-On Profile Data Base Structure

The actual profiles (lists of transactions and modes) are stored separately from the user IDs because frequently many users will want the same profile, and the data base administrator can avoid entering duplicate information.

CREATING THE SECURITY PROFILE

You must sign on to an application system yourself in order to create the sign-on authority for users. When IMSADF II is installed, the Sign-On Profile Data Base is already primed with authority for a user (such as a data base administrator) to sign on.

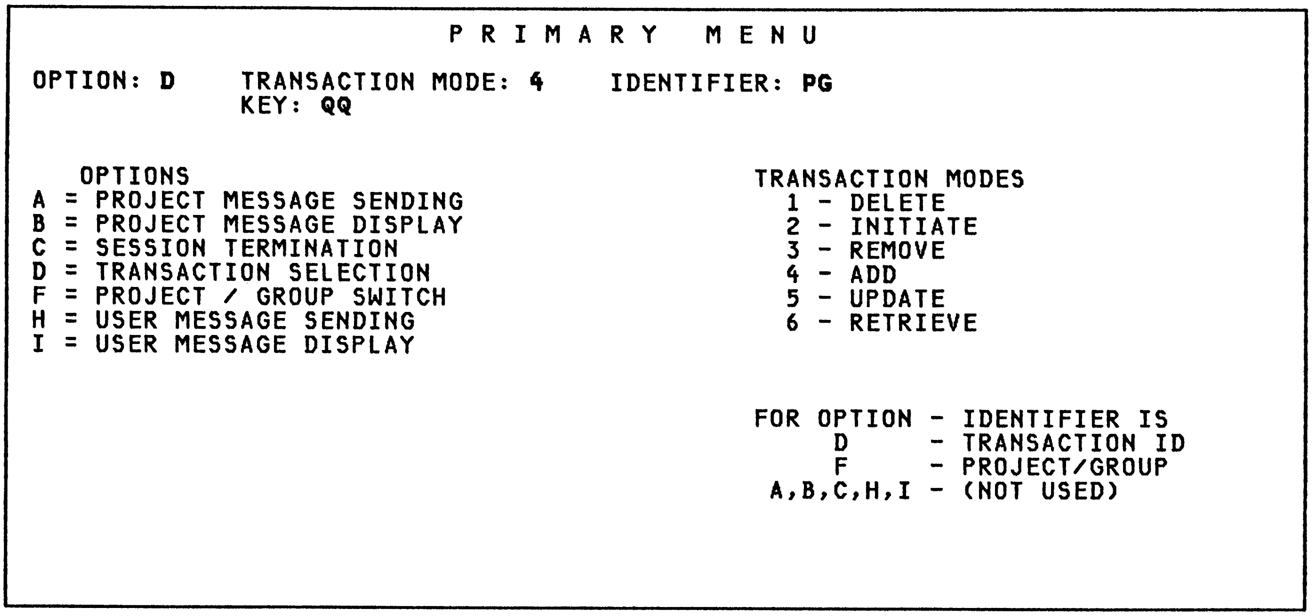
A variety of transactions are provided in the ???? application system (where ???? is the installed ADFID for which the default is MFC1). Three are concerned with security maintenance:

PG - maintains the PG root segment
SR - maintains the user ID segment (SR)
PR - maintains the profile segment (PR)

After you sign on, a Primary Option Menu is displayed, on which you enter:

```
  OPTION: D  TRANSACTION MODE: 4  IDENTIFIER: PG
             KEY: QQ
```

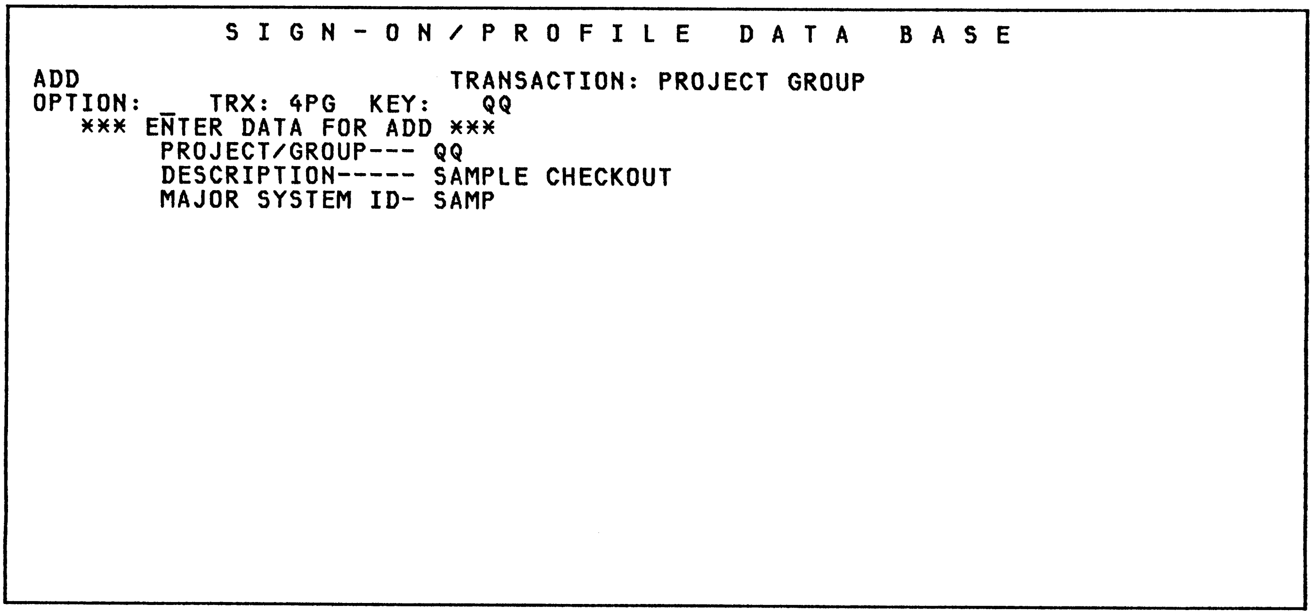
to add project/group QQ. (QQ is the key value of a PG segment.) See Figure 3-3.



```
                         P R I M A R Y   M E N U
OPTION: D    TRANSACTION MODE: 4    IDENTIFIER: PG
             KEY: QQ

   OPTIONS                                     TRANSACTION MODES
A = PROJECT MESSAGE SENDING                      1 - DELETE
B = PROJECT MESSAGE DISPLAY                      2 - INITIATE
C = SESSION TERMINATION                          3 - REMOVE
D = TRANSACTION SELECTION                        4 - ADD
F = PROJECT / GROUP SWITCH                       5 - UPDATE
H = USER MESSAGE SENDING                         6 - RETRIEVE
I = USER MESSAGE DISPLAY


                                               FOR OPTION - IDENTIFIER IS
                                                   D      - TRANSACTION ID
                                                   F      - PROJECT/GROUP
                                                A,B,C,H,I - (NOT USED)
```

Figure 3-3. Selection for Defining a New Project/Group

A Sign-On Profile Data Base screen appears (see Figure 3-4).



```
              S I G N - O N / P R O F I L E   D A T A   B A S E
ADD                        TRANSACTION: PROJECT GROUP
OPTION: _  TRX: 4PG  KEY:   QQ
   *** ENTER DATA FOR ADD ***
           PROJECT/GROUP--- QQ
           DESCRIPTION----- SAMPLE CHECKOUT
           MAJOR SYSTEM ID- SAMP
```

Figure 3-4. Defining a New Project/Group

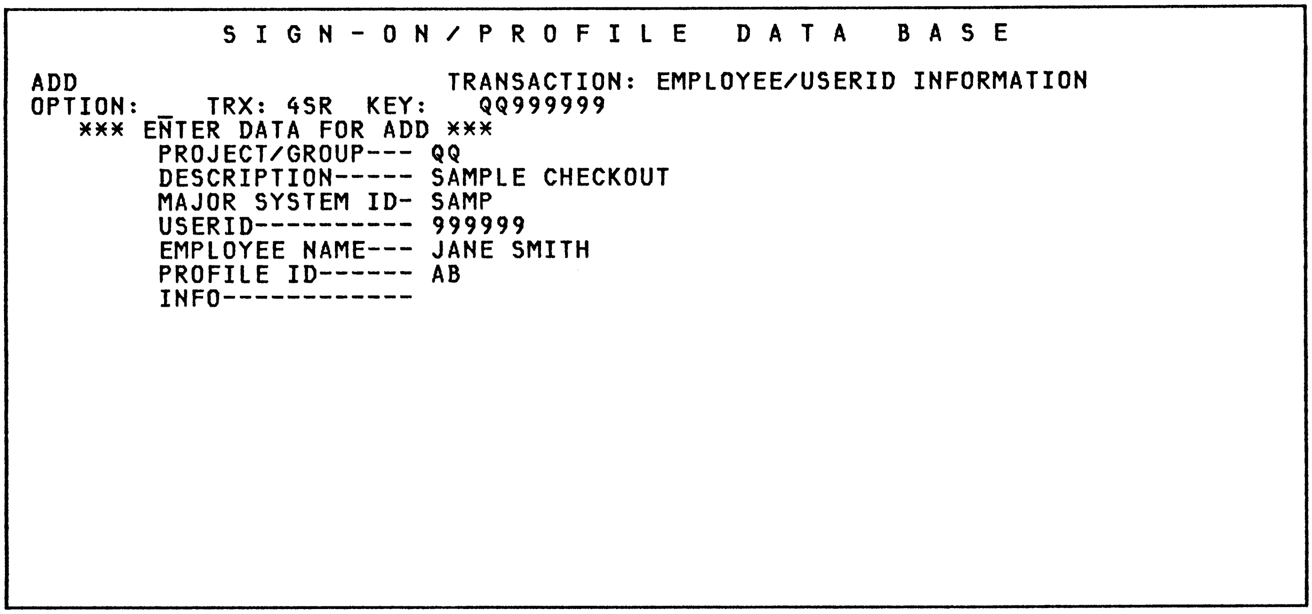
Enter the application system that project/group QQ will use, and, if desired, a descriptive name of that system. Remember, this code must be consistent with the PGROUP operand coded in the static rules on the Rules Generator SYSTEM or GENERATE statements. Otherwise, the project/group will not be able to use the transactions. (Abends with completion codes 806 will occur.)

Next, create the user ID. To do this, you must use transaction SR. To switch directly to the SR transaction without going through the menus, enter the following on the Sign-On Profile Data Base screen:

TRX: **4SR** KEY: **QQ999999**

This will define a user ID of 999999.

The screen in Figure 3-5 will appear.



```
               S I G N - O N / P R O F I L E   D A T A   B A S E

ADD                        TRANSACTION: EMPLOYEE/USERID INFORMATION
OPTION: _   TRX: 4SR  KEY:   QQ999999
   *** ENTER DATA FOR ADD ***
        PROJECT/GROUP--- QQ
        DESCRIPTION----- SAMPLE CHECKOUT
        MAJOR SYSTEM ID- SAMP
        USERID---------- 999999
        EMPLOYEE NAME--- JANE SMITH
        PROFILE ID------ AB
        INFO------------
```

Figure 3-5. Defining a User ID

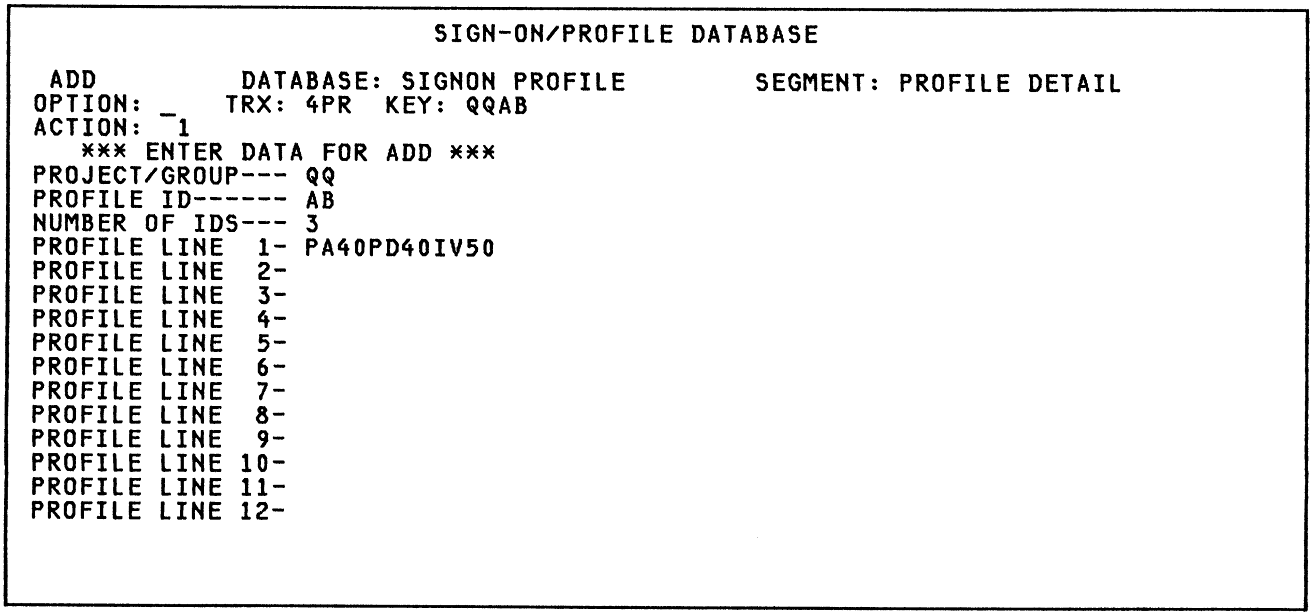
Enter the required PROFILE ID and, if desired, the EMPLOYEE NAME. You do not have to enter an actual list of authorized transaction IDs. INFO is also optional: it can be used to store lockwords, which are handled by the installation-defined lockword exit routine, if you use one (see Chapter 9, "Exits").

Finally, create the actual profile. On the Sign-On Profile DB Control screen enter:

TRX: **4PR** KEY: **QQAB**

to add a profile with ID **AB** under project/group QQ.

The screen in Figure 3-6 will appear.



```
                        SIGN-ON/PROFILE DATABASE
 ADD          DATABASE: SIGNON PROFILE        SEGMENT: PROFILE DETAIL
OPTION: _    TRX: 4PR  KEY: QQAB
ACTION:  1
   *** ENTER DATA FOR ADD ***
PROJECT/GROUP--- QQ
PROFILE ID------ AB
NUMBER OF IDS--- 3
PROFILE LINE  1- PA40PD40IV50
PROFILE LINE  2-
PROFILE LINE  3-
PROFILE LINE  4-
PROFILE LINE  5-
PROFILE LINE  6-
PROFILE LINE  7-
PROFILE LINE  8-
PROFILE LINE  9-
PROFILE LINE 10-
PROFILE LINE 11-
PROFILE LINE 12-
```

Figure 3-6. Creating a Profile

PROFILE LINE 1 contains the definition of a profile. Each entry is four characters long, of form:

XXLT

where:

XX is the transaction ID

L is the level of authority (the lowest processing mode allowed; for example, if L=5 then modes 5 and 6 are allowed). Possible levels are 3, 4, 5, and 6.

Note: Transaction modes 1 and 2 are interchangeable with modes 3 and 4, respectively. It is recommended that authority levels 3 and 4 be used rather than 1 and 2.

T controls whether the transaction ID is to appear on the Secondary Option Menu screen. Allowed values are:

T=0,1,2 or blank - show this transaction ID on the Secondary Option Menu screen

T=3 - do not show this transaction ID, but allow it to be selected through audit rules or special processing

Note: For a user in project/group AA to use these transactions, the Rules Generator must have been run with PGROUP=QQ coded on the SYSTEM statement or on the GENERATE statements for TRXID equal to PA, PD, and IV. If project/group ZZ is also allowed to use this application system, code extra GENERATE statements for the project/group ZZ.

The screen in Figure 3-6 has a second page of display for times when you need more than 12 lines of profile (more than 180 transaction IDs). The maximum number of transaction IDs in an application system is 300. To see the second page, press ENTER. The data will not be sent to IMSADF II until you press ENTER again. To return to the first page from the second, type **R1** in the ACTION field at the top of the screen and press ENTER. If you don't want to see the second page, then, after adding the necessary information on the first page, press PFKEY 4 if the terminal has program function keys. If the terminal does not have PF keys, type **E1** in the ACTION field at the top of the screen and press ENTER.

After creating the profile, type **C** in the OPTION field and press PFKEY 4 (or type **E1** in the ACTION field and press ENTER). The Primary Option Menu screen will return.

Now the security profile is complete and the application system can be tested.

USING BATCH INPUT OF DYNAMIC RULES

Dynamic rules can be entered in bulk using the batch processing capability of IMSADF II. You may find this method of entry more convenient when large numbers of rules must be coded. If the batch input is kept in step with your online changes, it can be used again to enter the rules into a production system when tests are complete.

Here is an example of batch input for a security profile:

```
//        EXEC   ????B
//TRANSIN DD   *
MFC1B3PGZZ
MFC1B4PGZZSAMPLE PROBLEM          SAMP
MFC1B4SRZZ999999PARTS USER AB
MFC1B4PRZZAB04
PA30PD30IV30CY30
/*
```

Note: **????** is the installed ADFID (the default is MFC1).

As can be seen, each batch transaction begins with transaction code:

ssssBmtx

where:

ssss is the application system ID (in this case, MFC1)
B is a literal
m is the transaction mode (1 to 5)
tx is the transaction ID.

Input records must be 80-byte card images, with the transaction code in column 1.

Note that whenever it is necessary to insert a root segment, it is deleted beforehand. In this way the deck can be re-run as often as necessary. On the first run, there will be error messages, since the segments will not be found. These can be ignored. A complete description of the batch transaction layouts is given below.

BATCH INPUT LAYOUTS

PG - Project/Group Segment

Card	Column	Length	Description
1	9	2	Project/group ID
	11	26	Description of PG function
	37	4	Application system ID

Sample:

```
MFC1B2PGQQSAMPLE CHECKOUT          SAMP
```

SR - Employee User ID Segment

Card	Column	Length	Description
1	9	2	Key of project/group segment
	11	6	Employee user ID
	17	11	Employee name
	28	2	Profile ID
	30	8	Information (optional lockword)

Sample:

```
MFC1B4SRQQ999999J.SMITH    AB
```

PR - Profile Authority Segment

Card	Column	Length	Description
1	9	2	Key of project/group segment
	11	2	Profile ID
	13	3	Number of transaction IDs

The following four positions are repeated 15 times per card (columns 1-60) on cards 2, 3, 4, 5, and 6.

1	1	2	Transaction ID
	3	1	Level of authority (1-6)
	4	1	Type of processing (0,1,2)
2	1	60	Transaction IDs and authority
3	1	60	
4	1	60	
5	1	60	
6	1	60	

Use the end of message characters to indicate end of data if fewer than 21 cards are specified. The end of message characters are defined at installation time (DEFADF). The default is $$.

Sample:

```
MFC1B4PRQQAB05
HD10SYS30PG10SR30PR30 $$
```

CHAPTER 4. THE AUDITOR AND THE AUDIT DATA BASE

The IMSADF II data bases can be in one of two formats. They can be either DL/I hierarchical data bases or DB2 relational data bases. This chapter describes the DL/I form of the Audit Data Base. Information about the maintenance of the DB2 data bases can be found in Chapter 4 of the IMS Application Development Facility II Version 2 Release 2 Application Development Reference.

Dynamic rules stored in the Audit Data Base are used to:

- Control validation of data field format and content
- Allow specification of calculation and logic operations
- manipulate keys and data
- Provide for additional security checking by key range or field values
- Support table definition and transaction switching

This data base can be maintained online using application system MFC1, or batch input can be used.

AUDITING FIELDS

Data validation, calculations and other processing against fields are performed by the Auditor, a common module that is a part of the transaction driver and is therefore included in all IMSADF II transactions. The operations it performs are controlled by rules stored in the Audit Data Base. In addition, certain operands must be coded on Rules Generator statements to request that audit operations be performed. If no such operands are present, the Auditor will simply validate the data entered by the user according to the data type coded on the Rules Generator FIELD statements. If errors are found, the fields in error are highlighted on the screen, and the user is invited to enter **E** to see the error messages (see Figure 1-10). Figure 4-1 illustrates the places the Auditor is called.

The Auditor can be called both during and after key selection, before the Data Display screen is shown to the user. Auditing that takes place during key selection is known as **key audit**. You can use a key audit to:

- Edit keys
- Cause a switch to another transaction based on the value of the key that the user has entered
- Validate keys and impose security by key range and user ID or terminal ID
- Alter or restrict the display of segments on the Secondary Key Selection screen (data base browsing)

If errors are detected during key audit, the keys in error are highlighted on the screen and the user is invited to enter **E** to display the error messages.

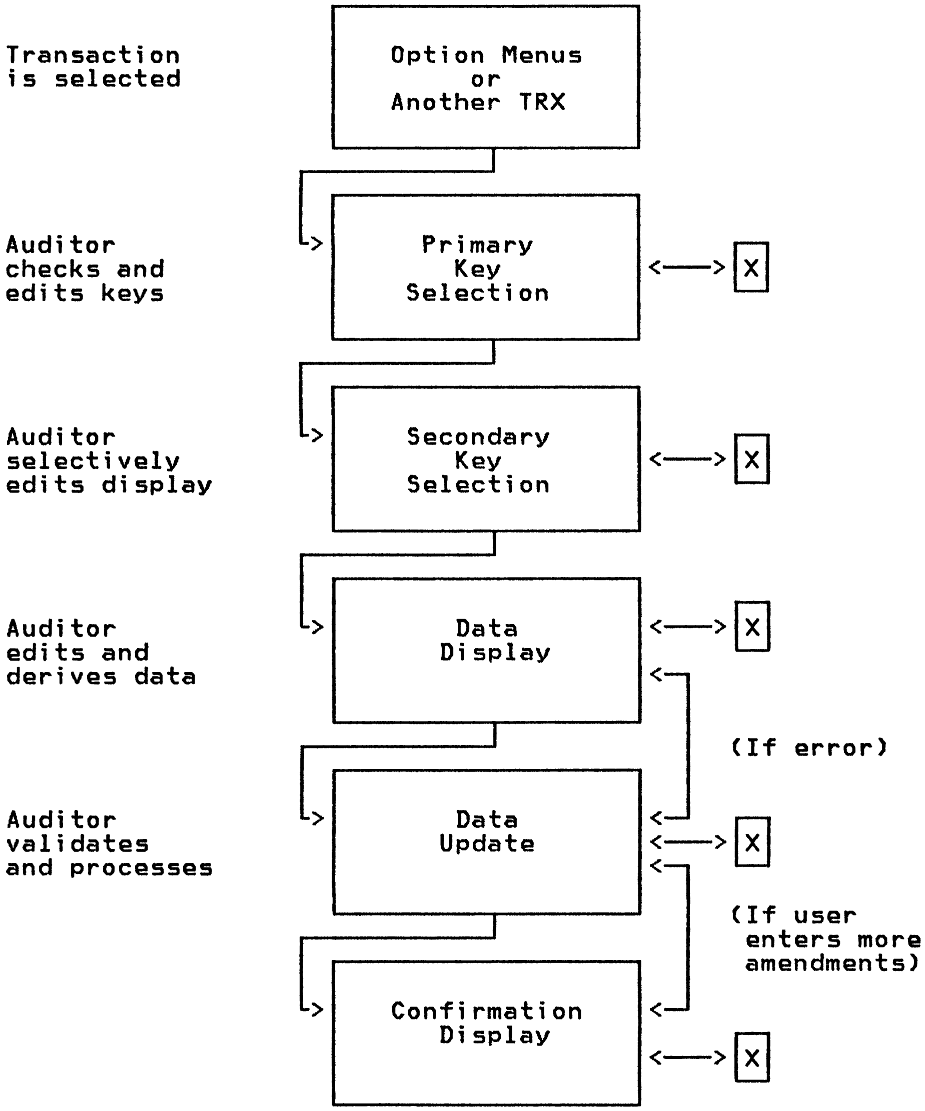

Transaction is selected
Option Menus or Another TRX
Auditor checks and edits keys
Primary Key Selection
X
Auditor selectively edits display
Secondary Key Selection
X
Auditor edits and derives data
Data Display
X
(If error)
Auditor validates and processes
Data Update
X
(If user enters more amendments)
Confirmation Display
X

Figure 4-1. Where the Auditor is Invoked

The next time the Auditor is called, is known as **preaudit**. This takes place after the DBPATH segments have been retrieved through key selection. Preaudit may be used to:

- Prevent some users from updating individual fields
- Convert certain data fields to a different format for viewing
- Initialize fields in a non-standard way
- Perform data base retrievals without using key selection for some or all of the segments

When errors or warnings are detected during preaudit, the user is prevented from viewing the Data Display screen. Instead, an Error Message screen appears, and the user must return to the Primary Key Selection screen without viewing the data.

The Auditor is called in transaction modes 1 to 5 after the user has viewed the Data Display screen and pressed ENTER and before the transaction driver issues DL/I calls to update the data bases. In transaction modes 1 to 4, the Auditor is called even if the user has not entered amendments. In transaction mode 5, the Auditor is called only if the user has entered amendments on the Data Display screen. If errors are found, the data base is not updated. When the user has entered corrections, the Auditor is called again. Several iterations can take place before the data bases are finally updated. If the user enters OPTION C (session termination) before clearing the errors, no updates are made.

After updates have been made successfully, the user may enter further amendments. The Auditor then validates and processes them and further data base updates can be performed.

The Auditor may be called in transaction mode 6 (Display) to carry out a transaction switch (see Chapter 6, "Complex Transactions") or to meet some unusual requirement. The PROCESS call of the Auditor will be invoked in transaction mode 6 if fields of MODE=4 are included in the transaction and if the user enters data into one or more of the MODE=4 fields.

REQUESTING AUDITS

The following operands on the Rules Generator FIELD statement determine when audit operations are performed against that field.

(or AU=Y) Auditing is performed if the field is changed.

(or FA=Y) Forces the field to be audited by marking it as changed. AUDIT=Y must also be coded for this to take effect.

(or PA=Y) Requests auditing on the preaudit pass.

REQ=Y The field must not be an initialized value. Initialized values are either blanks or zeros depending upon the field type. The user will be required to enter a non-initialized value.

(or AS=YFPR) Equivalent to coding all four of the above operands. Select from the values: Y-AUDIT, F-FAUDIT, P-PAUDIT, R-REQ. Do not mix ASTATUS and the other operands (the others will be ignored when ASTATUS is present).

(or KA=YES) Requests auditing during primary and secondary key selection.

(or KA=PRIM) Requests auditing during primary key selection.

(or KA=SECO) Requests auditing during secondary key selection.

THE AUDIT DATA BASE

The dynamic rules that control Auditor operations are stored in the Audit Data Base or in the Static Audit Load Modules. Figure 4-2 shows its structure, with the names of the segments in the field audit leg. (The segments in the other legs will be considered later).

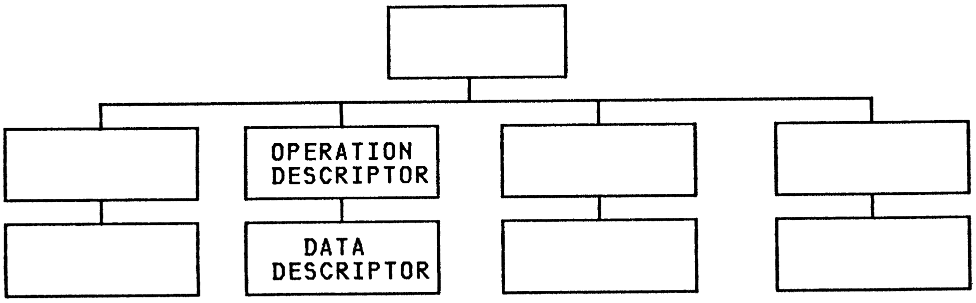


Figure 4-2. Audit Data Base -- Field Audit Leg

The operations to be performed against a data field are described in one or more operation descriptor segments; literal values are held in data descriptor segments. The root segment is an anchor point and contains only a key, which is based on the name of the field to be audited.

The eight-character field name is:

SSXXFFFF

where:

```
  SS  is the first two characters of the application system ID
  XX  is the segment ID
FFFF  is the field ID
```

The key of the root segment has the form:

SSSSAAAASSXXFFFF

where:

```
    SSSS  is the application system ID
    AAAA  is the audit group code
SSXXFFFF  is the field name
```

The audit group code is used to request different sets of audit operations against the same field in different transactions or for different project/groups. In many cases, such separation is not necessary and a default value of YYYY is used throughout the system. If multiple audit group codes are needed, the AGROUP operand is added to the GENERATE statements for the transactions that require it.

AUDIT OPERATIONS

The Auditor performs 12 types of operations:

- Comparisons
- Range and list checks
- Type checks
- Assignments (by field or segment)
- Arithmetic
- Checks and settings of control information
- DL/I calls
- SQL calls
- Transaction switching
- Table look-up
- Subroutine calls
- Flow of control (iteration, GOTO)

The Auditor reads the rules in the Audit Data Base to determine what to do. The operation instructions are stored in the operation descriptor segments of the Audit Data Base. The layout is shown in Figure 4-3.

2	2	8	2	2	4
Key Field	Operation Code	Related Field	Next True	Next False	Message Number

Figure 4-3. Operation Descriptor Segment Layout

Each operation descriptor segment has a **key field**, which is a two-character sequence code. The first key field will normally be 01 or AA.

The **operation code** is either one of a list of standard codes (see the IMS Application Development Facility II Version 2 Release 2 Application Development Reference) or a user code designating an operation by a user-written exit routine in COBOL, PL/I or Assembler.

Many operations require a second field, which is used to compare with the audited field, to assign a value, to store intermediate results, or for some other reason. This **related field** must be in a segment defined to the transaction via the GENERATE statement. Related fields can be pseudo segment or target segment fields. Fields in segments above target segments in the same hierarchical paths can be related fields provided the segments are included in the transaction. Such segments are included in the transaction if they have at least one displayable field. If such a segment has no displayed field but is required for auditing, it can still be included by naming it explicitly in the DBPATH operand of the GENERATE statement for the transaction. The Rules Generator includes it in the transaction without making it a target segment.

Several audit operations can be coded against one field. Each operation is stored in an operation descriptor segment with a different key field sequence number. As soon as one operation is complete, the Auditor examines the **next true** and **next false** contents to decide what to do next. These contain the key field of the next operation descriptor to be performed. There is a logical branching capability. Many operations return a true or false indicator which determines which of two key sequence numbers to branch to next. A data comparison will be true if the field is equal to the related field and false otherwise. If the fields are equal, the Auditor will branch to the next true key sequence number, which may be behind or in front of the present one.

The value 00 in the next true or false positions tells the Auditor that validation on this field is complete, and it can start on the next field marked for audit.

Within one segment, fields are audited in the order in which they are coded in the Rules Generator. If the IMS/VS DB sequence field is divided into several IMSADF II key fields, then the IMSADF II key fields must be coded in DB order. Within a transaction, segments are audited beginning at the highest level in the data base and working down each hierarchical path. Paths are audited in the order in which their target segments appear in the DBPATH operand of the GENERATE statement. Pseudo segments are audited before data base segments, in the order in which they appear in the TSEGS operand.

If processing for every audited field leads to a 00 next true or false condition, the transaction can be completed.

Audit rules indicate errors by a blank next true or false position, followed by a four-digit error **message number**. The audited field will be redisplayed and highlighted. The message number is a reference to the actual text, which is coded separately and stored in the Message Data Base.

THE HIGH LEVEL AUDIT LANGUAGE

A compiler is provided to generate the audit rules in the appropriate format for storage in the Audit Data Base. The input to the compiler is a series of statements in a high level audit language.

This language is somewhat like PL/I, although there are a great many differences. The overall structure is inherited from the organization of the Audit Data Base, with its use of data descriptor segments to hold literal values and its separation of processing applicable to different fields.

For that reason, each section of code - or program - is headed by the FIELD statement. This gives the name of the audited field to which the statements that follow will apply. The first FIELD statement submitted to the compiler must be preceded by SYSID and SEGID statements giving the application system ID and segment ID in which the audited field occurs.

For example:

```
SYSID = SAMP
SEGID = IV
FIELD = STCK
```

would precede the definition of the Auditor processing against field STCK in segment IV in application system SAMP. If the field MSTK in the same segment were to be audited as well, the statements defining that processing need only be preceded by:

```
FIELD = MSTK
```

In effect, these statements -- called **headers** -- define the key in the Audit Data Base under which the rules are to be stored. In this case, the generated key would be SAMPYYYYSAIVSTCK. If an audit group code other than YYYY is used in the input to the Rules Generator, add:

```
AGROUP = AAAA
```

after the SYSID statement, where AAAA is the required audit group code.

Next, it is necessary to distinguish among the four calls of the Auditor:

- Key audit (the KEY call)
- Preaudit (the PRELIM call)
- Audit (the PROCESS call)
- Tables (TABLES creation)

This is done by means of another header -- KEY, PRELIM, PROCESS, or TABLES -- following the FIELD statement. Finally, referring to Figure 4-2, there are three phases of the Audit Data Base into which the operation descriptors can go. The field audit phase is identified in the high level language as P1.

Hence, the complete header information for audit processing against the STCK field would be as follows:

```
SYSID = SAMP
SEGID = IV
FIELD = STCK
PROCESS
P1
```

After the header information, actual validation and processing requirements must be specified. For a full description of the capabilities of the high level audit language compiler, refer to the IMS Application Development Facility II Version 2 Release 2 Application Development Reference.

BASIC GUIDELINES FOR CODING IN THE HIGH LEVEL AUDIT LANGUAGE

- Fields are referenced by their full eight-character names (of form SSXXFFFF), except for the audited field (the field named in the FIELD statement), which need only be referenced with the four-character field ID.

- Assignments and arithmetic operations are requested as illustrated by these examples:

```
STCK = SAIVOSTK + STCK
SAIVMSTK = STCK / 33.33
SAPADESC = 'LEFT HANDED WIDGET'
```

- No nested or parenthesized expressions can be accepted. Only one arithmetic operation per assignment statement is allowed.

- Quoted literals may contain embedded quotes, each represented by two quotation marks, but may not contain commas or right parentheses.

- The syntax of the IF statement is illustrated by this example:

```
IF STCK > 50
  SAIVMSTK = STCK * 1.5
ELSE
  SAIVMSTK = SAIVOSTK - 2
ENDIF
```

- To request that an error message to be sent to the user, code:

```
ERRORMSG = nnnn
```

  where **nnnn** is the four-digit error message number. The audited field (i.e., that named in the preceding FIELD statement) will be marked in error and highlighted on the screen.

  To cause another field to be so marked, code:

```
SETERRMSG SSXXFFFF=nnnn
```

  where **SSXXFFFF** is the name of the field to be marked in error.

  After the ERRORMSG statement is executed, auditing of the field terminates; after the SETERRMSG statement, processing continues with the next statement.

- To terminate auditing for a field without error, code the statement EXIT.

- All tokens, whether names, literals or operations, must be separated by spaces. Thus, equals signs and arithmetic operations must have a blank space on each side.

Example

The field STCK must be less than or equal to MSTK and more than OSTK. The Rules Generator statements are:

```
FIELD     ID=STCK,TYPE=DEC,LENGTH=5,AUDIT=YES
FIELD     ID=OSTK,TYPE=PD,LENGTH=3
FIELD     ID=MSTK, TYPE=DEC,LENGTH=5
```

The high level audit language coding is:

```
  SYSID = SAMP
  SEGID = IV
  FIELD = STCK
  PROCESS
  P1
  IF STCK > SAIVMSTK
    ERRORMSG = 9224
  ENDIF
  IF STCK <= SAIVOSTK
    ERRORMSG = 9224
  ENDIF
```

The operation descriptors that will be generated by the compiler for placement in the Audit Data Base are as follows:

```
  Audit root key:                       SAMPYYYYSAIVSTCK
  Audit operation descriptors:          0102SAIVMSTK  029224
                                        0202SAIVOSTK00  9224
```

The fields are assumed to be in the IV segment in the SAMP application. Audit operation code 02 returns true if the audited field is greater than the related field. Error message 9224 will say: "Invalid stock level amendment."

DATA DESCRIPTORS

Some audit operations require data values. These are coded as numerals or as quoted alphanumeric literals. They are stored in data descriptor segments beneath the relevant operation descriptor. For example, if it is necessary to make sure that a field is in a constant range, code:

```
  FIELD = STCK
  PROCESS
  P1
  IF STCK NOT IN 900:1035
    ERRORMSG = 9225
  ENDIF
```

This will result in the following operation and data descriptors being generated by the compiler:

```
  Operation descriptor:  0121         00   9225
  Data descriptor:       0001(900,1035)
```

Figure 4-4 shows the data descriptor format. If many values are needed, multiple data descriptors can be created. The Auditor will convert the data values to the data type of the audited or related field with decimal point alignment and padding as appropriate.

4	24
Key Sequence number	Data Values (Value, value,...value)

Figure 4-4. Data Descriptor Segment Layout

ADDITIONAL CAPABILITIES OF THE AUDITOR

Other important capabilities of the Auditor are described below.

CONTROL INFORMATION

By using the names reserved, it is possible to test and set system information such as the logical terminal name, the user ID and project/group currently signed on to the application, the application system ID, and the transaction mode and ID. The reserved names are LTERM, USERID, PGROUP, SYSID, MODE, and TRXID, respectively.

The attributes of a displayed field can also be set dynamically. For example, you can set the STCK field in the IV segment to be highlighted and nonmodifiable when the screen is displayed by coding PAUDIT=YES on the Rules Generator FIELD statement and writing the following high level audit language code:

```
  SEGID = IV
  FIELD = STCK
  PRELIM
  P1
  STCK HILITE = ON
  STCK UPDATE = OFF
```

It is possible to reposition the cursor dynamically, to set a field as premodified (will be read in from the screen even if the user does not change it), and to mark it as changed (causes the segment to be updated). The respective keywords in the language are CURSOR, PREMODIFY, and CHANGED. Setting a related field changed will cause the related field to be audited if it is marked AUDIT=YES and occurs later in the transaction than the field being audited.

For color terminals, colors and extended highlighting can be set using the keywords COLOR and XHILT. To make the STCK field blink in red, code:

```
    STCK  COLOR  =  RED
    STCK  XHILT  =  BLINK
```

Allowed colors are PINK, BLUE, GREEN, RED, WHITE, YELLOW, and TURQUOISE. Allowed extended highlighting options are UNDERSCORE, REVERSE, BLINK, and DEFAULT (i.e., no highlighting).

DL/I CALLS

DL/I calls have a number of uses in auditing. One is to validate that a value entered by a user is a key in a data base; another is to retrieve multiple segment occurrences (twins) (see Chapter 6, "Complex Transactions").

For example:

```
  IF GU KEYFIELD IV NOT OK
    IF STATCODE NOT = 'GE,GB'
      ROLLCALL = 'AN UNEXPECTED ERROR HAS OCCURRED. CONTACT SYSTEM SUPPORT'
    ENDIF
  ENDIF
```

This will retrieve the IV segment with a DL/I call of GU (Get Unique). In the event of failure, the DL/I status code returned can be examined by using the special system name STATCODE. The status codes GE and GB are normal "not found" conditions. If an abnormal condition has arisen, the error message given will be sent to the terminal user; any data base updates performed since the time the user entered the screen will be undone, and the transaction will be terminated.

If segments are to be retrieved in this way, they must be coded in the Rules Generator input and named in the TSEGS operand of the GENERATE statement for the transaction.

TABLE HANDLING

Figure 4-5 shows a typical table consisting of multiple rows and two columns.

Argument (1-8 characters)	Value (1-70 characters)
B	BELGIUM
CH	SWITZERLAND
D	GERMANY
DK	DENMARK
E	SPAIN
F	FRANCE
GB	GREAT BRITAIN
I	ITALY
N	NORWAY
NL	NETHERLANDS
S	SWEDEN
SF	FINLAND
USA	UNITED STATES

Figure 4-5. A Table Named COUNTR

Tables have six-character names. They must be stored in the Audit Data Base separately from the operation descriptors under special root segment keys, as shown in Figure 4-6. Under one root segment (e.g., with key CENTRALALLTABLES), many different tables can be held. Therefore, the fully qualified name of the table COUNTR would be CENTRALALLTABLESCOUNTR in the example (22 characters in all). Eight operations can be performed. Reference the appropriate table by quoting either the 22-character full table name or a field containing the name in the high level audit language statements. Encode/decode operations can be performed, as well as table lookup.

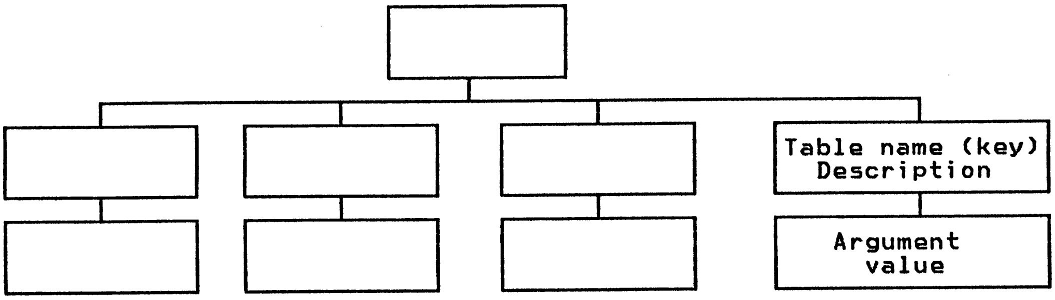


Figure 4-6. Audit Data Base Storage of Tables

In the example of country codes, the field CODE (PAUDIT=YES) in the segment DA is to be decoded and the name of the country placed in field CNAM in segment PS:

```
SEGID = DA
FIELD = CODE
PRELIM
P1
IF DECODED CODE TO SAPSCNAM USING 'CENTRALALLTABLESCOUNTR' OK
  NOP
ENDIF
```

NOP means no operation. In this example we will take no special action if the code is not in the table. It will simply appear blank.

SUBROUTINE CALLS

Sometimes several fields require similar series of operations to be performed against them, and it is convenient to write the operation descriptors and data descriptors once for all of them. This can be done by placing them under a separate root segment key in the Audit Data Base and branching to them. They thus constitute a subroutine. The name of the subroutine is the 16-byte key of the root segment under which it is held; it may be in any form. The SUBNAME statement heads a subroutine. Subroutines are called through the CALL statement.

Suppose that the subroutine is named SAMPYYYYEDITDATE. It will be headed:

```
SUBNAME = 'SAMPYYYYEDITDATE'
```

It should use the same headers (KEY, PRELIM, PROCESS and P1, etc.) to distinguish the different phases of the Auditor. These headers come after the SUBNAME statement. The call statement would be:

```
CALL 'SAMPYYYYEDITDATE'
```

The only parameter that can be passed is the audited field.

EXAMPLES OF AUDITING (APPLICATION SYSTEM SAMP)

- On the preaudit pass, limit user 172467 to access of part numbers beginning with 025 to 999.

  Significant Rules Generator statements:

  ```
  SEGMENT     ID=PA,...
  FIELD       ID=KEY,PAUDIT=Y,LENGTH=17,....
  ```

  High level audit language:

  ```
  SYSID = SAMP
  SEGID = PA
  FIELD = KEY
  PRELIM
  P1
  IF USERID = '172467'
    IF KEY < '025'
      ERRORMSG = 0001
    ENDIF
  ENDIF
  ```

  Generated segments:

  ```
  Root key:    SAMPYYYYSAPAKEY
  Audit logic: 0116        0200        Is this pre-audit?

               0268        0300        Is this 172467?
                 0001(172467)

               0304        00  0001    Number <025?
                 0001(025)
  ```

- Add change to existing stock; if negative, send error message.

  Significant Rules Generator statements:

```
SEGMENT ID = IV,...
FIELD   ID = CHA,TYPE=PD,LENGTH=5,DEC=2,AUDIT=YES
FIELD   ID = STCK,TYPE=PD,LENGTH=5,DEC=2
```

  High level audit language:

```
SYSID = SAMP
SEGID = IV
FIELD = CHA
PROCESS
P1
ACCUM = ACCUM + CHA
ACCUM = ACCUM + SAIVSTCK
IF ACCUM < 0
 ERRORMSG = 0002
ELSE
SAIVSTCK = ACCUM
ENDIF
```

  Generated segments:

```
Root key:       SAMPYYYYSAIVCHA
Audit logic:    0150        02          Add CHA to ACCUM

                0251SAIVSTCK03          Add STCK to ACCUM

                03A5          040002    Error if 0 > ACCUM
                  0001(0)

                0463SAIVSTCK00          Move total to STCK
```

CREATING AND MAINTAINING AUDIT RULES

Transactions are provided in the application system MFC1 to define and amend each individual segment type in the Audit Data Base. The IDs are shown in Figure 4-7. Any changes made this way will not be reflected in the high level audit language statements that may have been written. These must be altered separately.

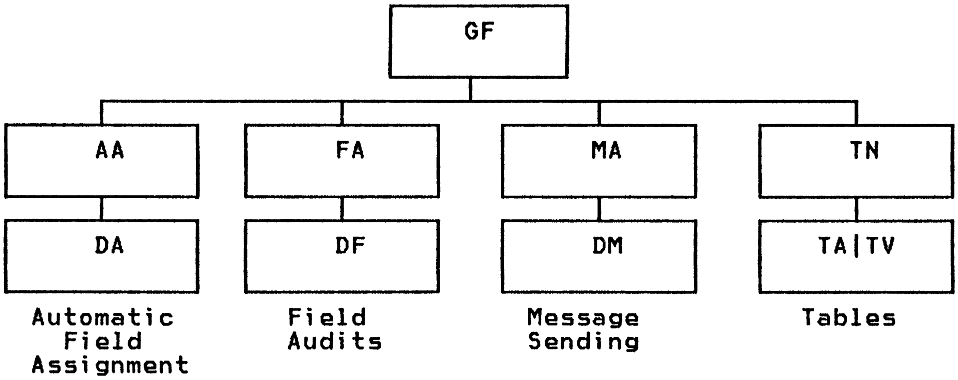
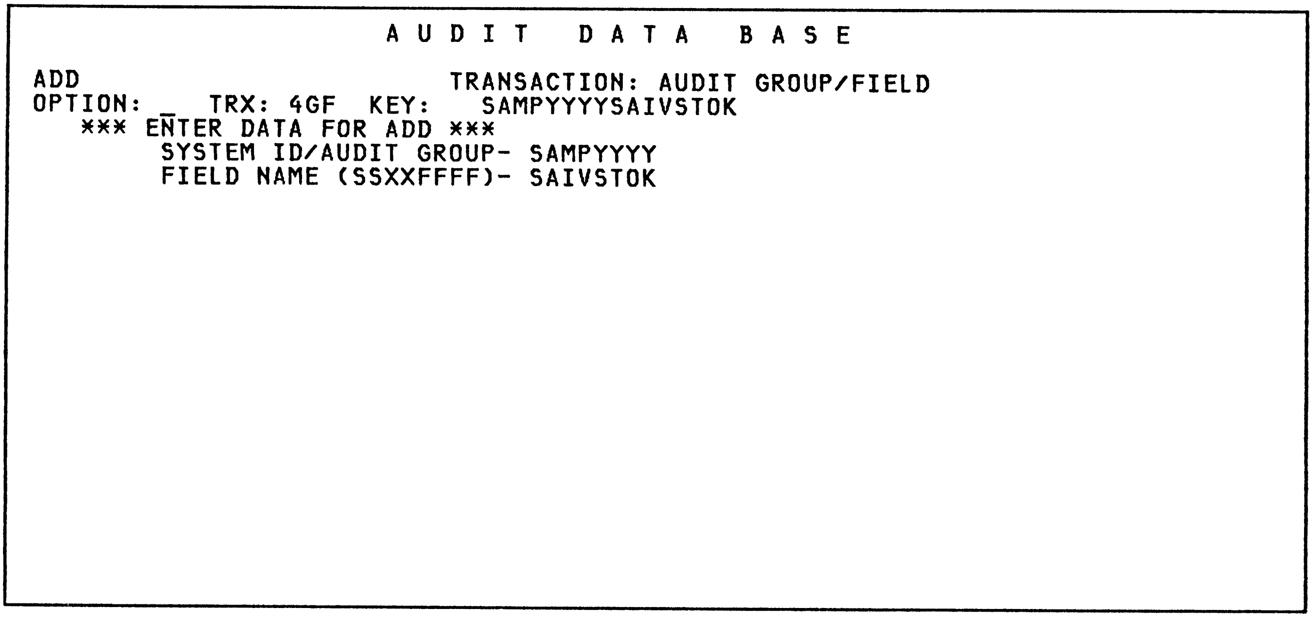


Figure 4-7. Segments and Corresponding Transaction IDs

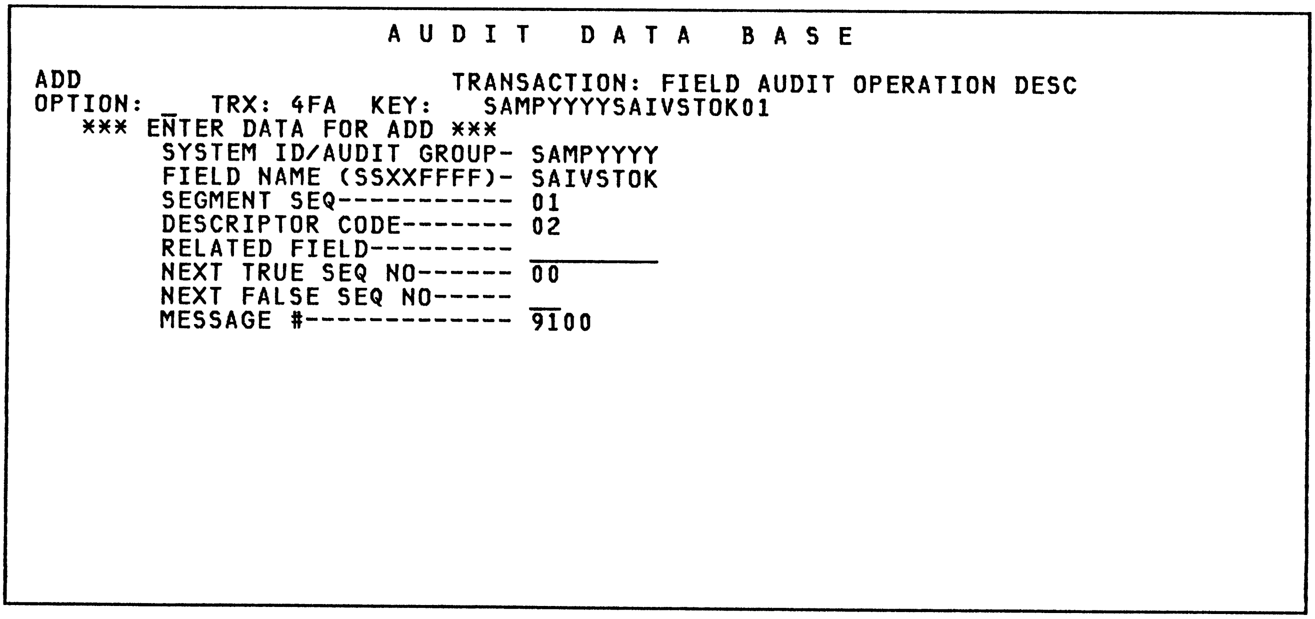
First, the GF transaction is invoked to create a root segment, as shown in Figure 4-8.



```
                          A U D I T   D A T A   B A S E

ADD                        TRANSACTION: AUDIT GROUP/FIELD
OPTION: _  TRX: 4GF  KEY:   SAMPYYYYSAIVSTOK
   *** ENTER DATA FOR ADD ***
        SYSTEM ID/AUDIT GROUP- SAMPYYYY
        FIELD NAME (SSXXFFFF)- SAIVSTOK
```

Figure 4-8. Inserting a Root Segment into the Audit Data Base

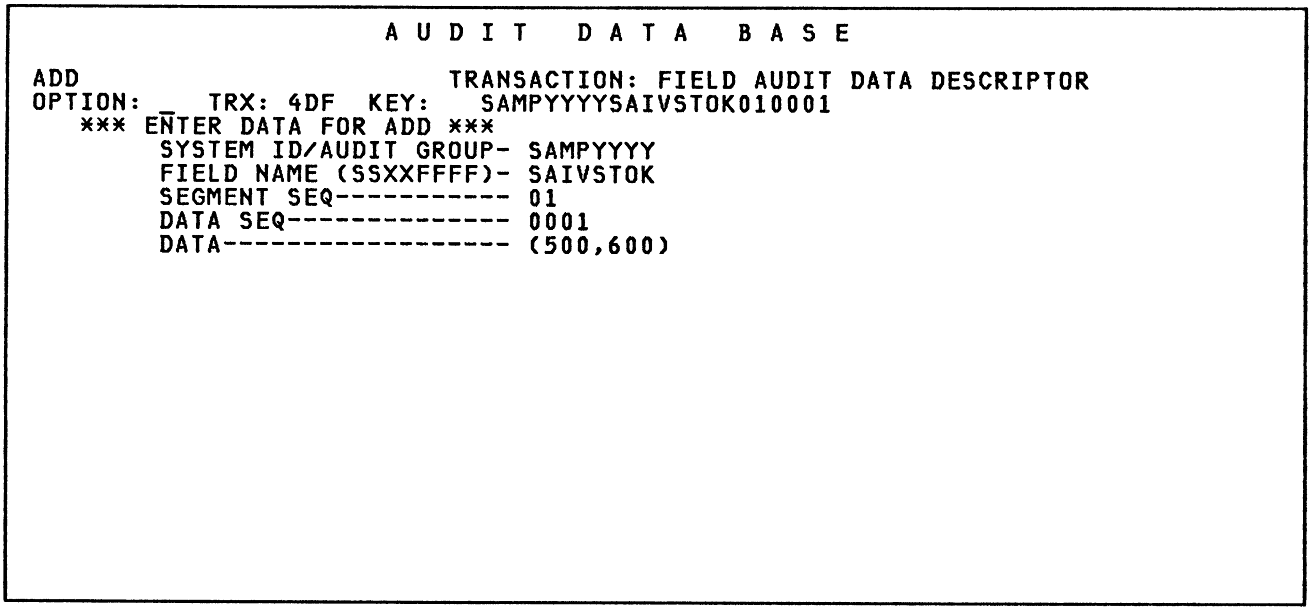
To add an operation descriptor for a field audit, change the TRX value to 4FA and append the key value 01. In Figure 4-9, a range check is being requested.



```
                          A U D I T   D A T A   B A S E

ADD                        TRANSACTION: FIELD AUDIT OPERATION DESC
OPTION: _  TRX: 4FA  KEY:   SAMPYYYYSAIVSTOK01
   *** ENTER DATA FOR ADD ***
        SYSTEM ID/AUDIT GROUP- SAMPYYYY
        FIELD NAME (SSXXFFFF)- SAIVSTOK
        SEGMENT SEQ----------- 01
        DESCRIPTOR CODE------- 02
        RELATED FIELD--------- ________
        NEXT TRUE SEQ NO------ 00
        NEXT FALSE SEQ NO----- __
        MESSAGE #------------- 9100
```

Figure 4-9. Defining an Operation Descriptor

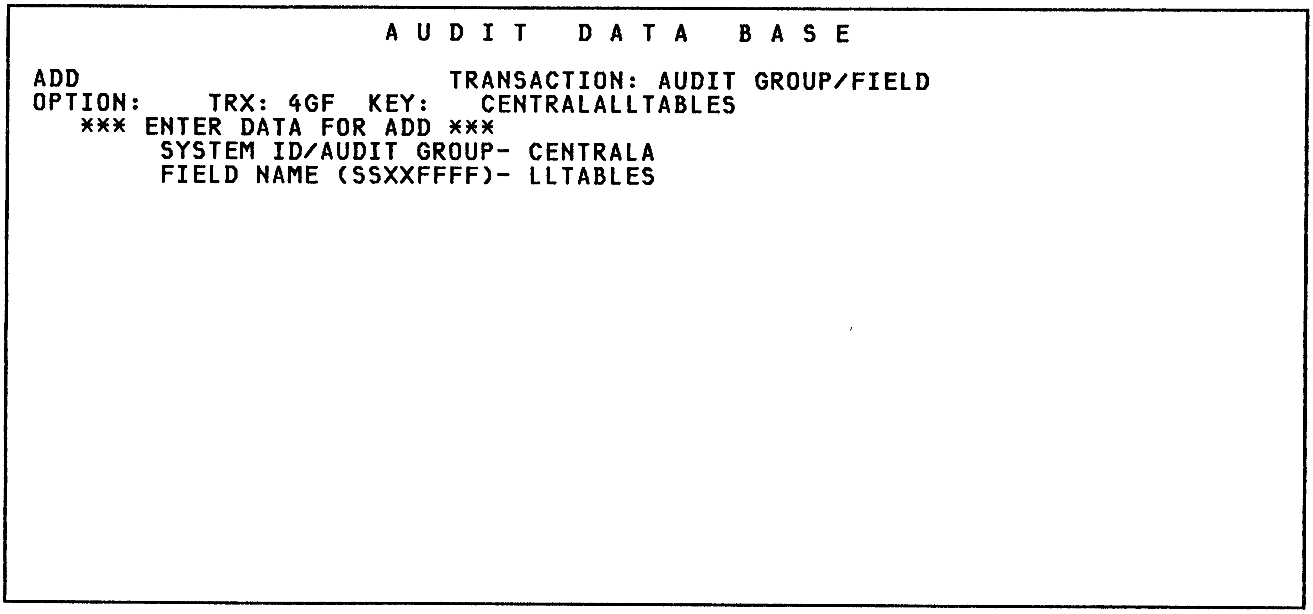
To define the data descriptor, change the TRX value to 4DF and append 0001 to the concatenated key to receive the display shown in Figure 4-10 on which the range values are entered.



```
                    A U D I T   D A T A   B A S E

ADD                        TRANSACTION: FIELD AUDIT DATA DESCRIPTOR
OPTION: _  TRX: 4DF  KEY:   SAMPYYYYSAIVSTOK010001
   *** ENTER DATA FOR ADD ***
        SYSTEM ID/AUDIT GROUP- SAMPYYYY
        FIELD NAME (SSXXFFFF)- SAIVSTOK
        SEGMENT SEQ----------- 01
        DATA SEQ-------------- 0001
        DATA------------------ (500,600)
```

Figure 4-10. Defining a Data Descriptor

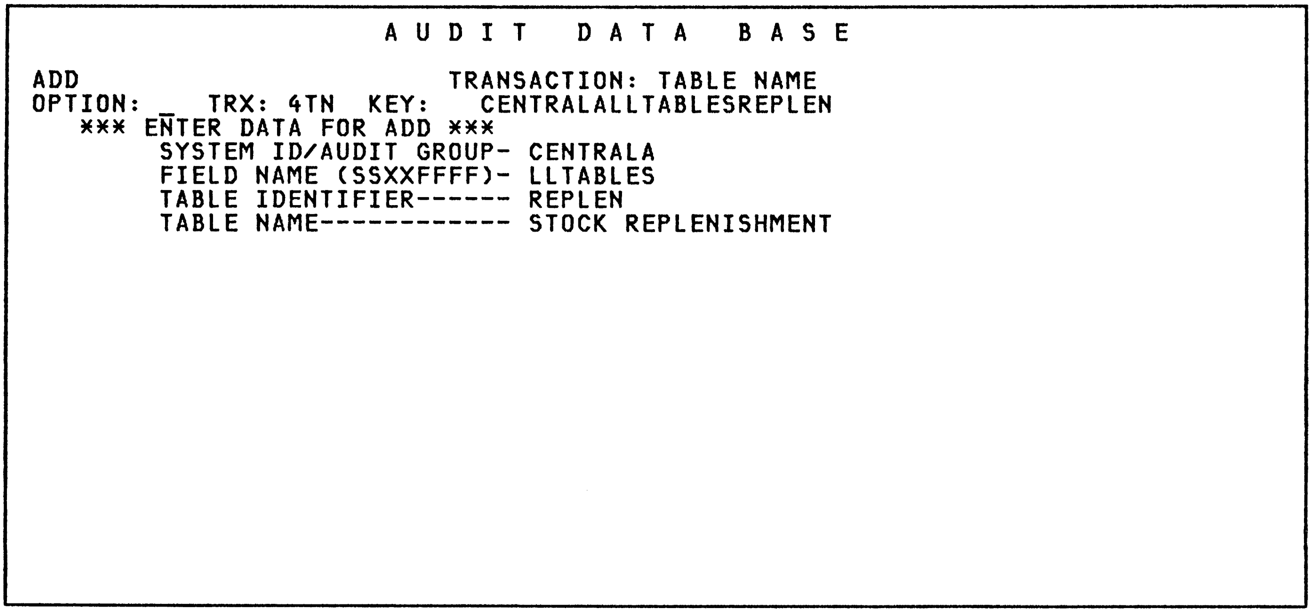
To set up a table, a root segment under which to store tables must first be created, as in Figure 4-11.



```
                    A U D I T   D A T A   B A S E

ADD                        TRANSACTION: AUDIT GROUP/FIELD
OPTION:    TRX: 4GF  KEY:   CENTRALALLTABLES
   *** ENTER DATA FOR ADD ***
        SYSTEM ID/AUDIT GROUP- CENTRALA
        FIELD NAME (SSXXFFFF)- LLTABLES
```

Figure 4-11. Inserting a Root Segment in the Audit Data Base in Readiness to Define Tables

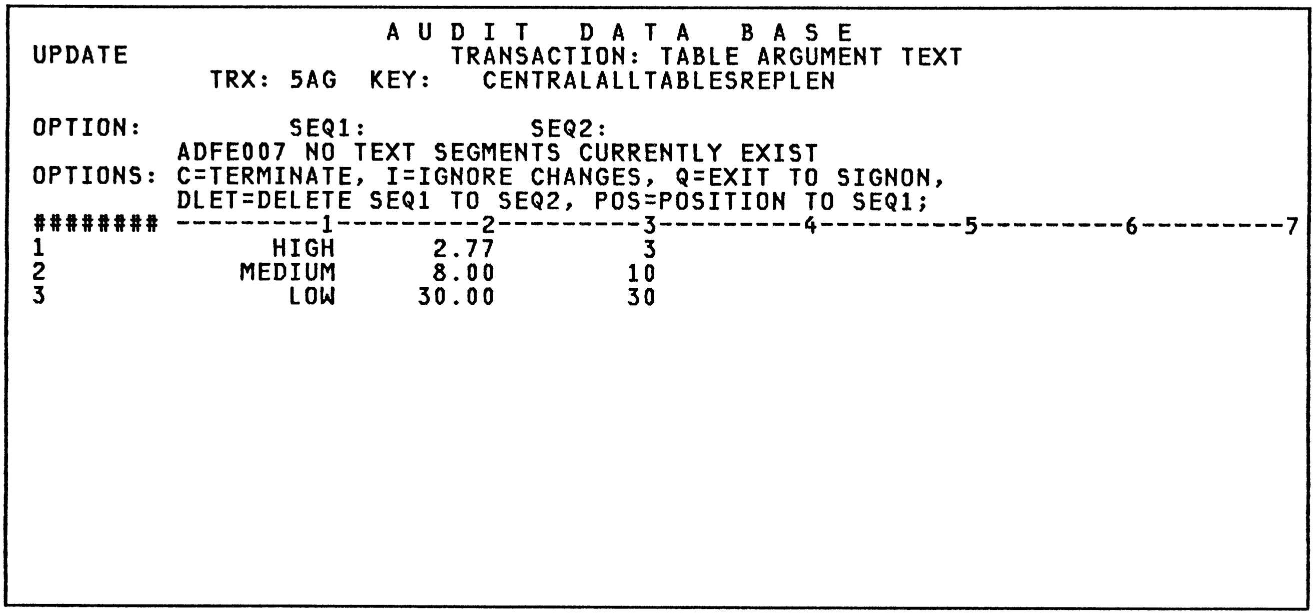
To enter a table name segment, change the TRX value to 4TN and append the six-character table name REPLEN to the 16-character root key (see Figure 4-12).



```
                         A U D I T   D A T A   B A S E

ADD                          TRANSACTION: TABLE NAME
OPTION: _   TRX: 4TN  KEY:   CENTRALALLTABLESREPLEN
   *** ENTER DATA FOR ADD ***
        SYSTEM ID/AUDIT GROUP- CENTRALA
        FIELD NAME (SSXXFFFF)- LLTABLES
        TABLE IDENTIFIER------ REPLEN
        TABLE NAME------------ STOCK REPLENISHMENT
```

Figure 4-12. Defining a Table Name

Now alter the TRX value to 5AG. This leads to a text editing screen (see Figure 4-13) on which actual table entries are entered.



```
                         A U D I T   D A T A   B A S E
UPDATE                       TRANSACTION: TABLE ARGUMENT TEXT
           TRX: 5AG  KEY:   CENTRALALLTABLESREPLEN

OPTION:         SEQ1:           SEQ2:
         ADFE007 NO TEXT SEGMENTS CURRENTLY EXIST
OPTIONS: C=TERMINATE, I=IGNORE CHANGES, Q=EXIT TO SIGNON,
         DLET=DELETE SEQ1 TO SEQ2, POS=POSITION TO SEQ1;
######## ----------1----------2----------3----------4----------5----------6----------7
1              HIGH      2.77         3
2            MEDIUM      8.00        10
3               LOW     30.00        30
```

Figure 4-13. Creating Table Entries (One Line Per Entry)

The one-character codes (1, 2, and 3) are the table arguments in this example. The values each appear to consist of three separate items, but to the Auditor they are a single character string. If you want to treat them as separate values, you can define a pseudo segment.

For example:

```
  SEGMENT  ID=TW,TYPE=PS,DISP=NO
  FIELD    ID=DESC,LENGTH=10,DISP=YES
  FIELD    ID=THRS,LENGTH=10
  FIELD    ID=ORDQ,LENGTH=10
  FIELD    ID=FULL,LENGTH=30,PAUDIT=Y,POS=1
  FIELD    ID=THRP,LENGTH=5,DEC=2,TYPE=PD,PAUDIT=Y
  FIELD    ID=ORQP,LENGTH=5,TYPE=PD,PAUDIT=Y
```

Assuming that the type of inventory is a one-byte code in field INVC in segment PD in system SAMP, the Auditor coding to retrieve the correct table entry (and display the account description) and convert the numbers to a form in which they can be used for arithmetic is as follows:

```
  SYSID = SAMP
  SEGID = TW
  FIELD = FULL
  PRELIM
  P1
  IF DECODED SAPDINVC TO FULL USING 'CENTRALALLTABLESREPLEN' NOT OK
    ERRORMSG = 0604
  ENDIF
  * Convert THRS and ORDQ (character) to THRP and ORDP
  SATWTHRP = SATWTHRS
  SATWORDP = SATWORDQ
```

ERROR MESSAGES

As all the examples have shown, messages are numbered. In fact, the four-digit numbers are unique within each application system. The full identifier of a message is:

ssssnnnn

where:

ssss is the application system ID
nnnn is the message number

Messages consist of the text to be displayed to the user when audit errors occur, together with a list of field names in the transaction when it is desired to show data values as well as literal text.

They are stored in the Message Data Base, which is one of the three dynamic rules data bases. As shown in Figure 4-14, the message header and message text segments are used to store error messages. Transactions HD and SY are used to create and maintain each segment type.

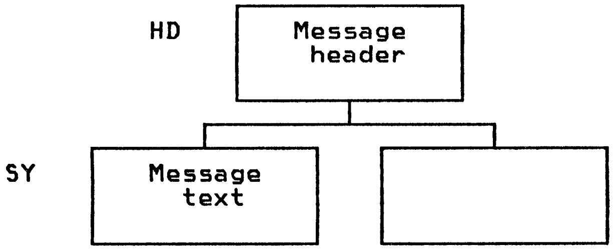


Figure 4-14. Error Messages in the Message Data Base

Figure 4-15 depicts the layout of these segments. An error message can be up to 980 characters in length and occupy from 1 to 14 message text segments. Message text sequence numbers begin with 00000001. Messages of 70 characters or less need only one segment.

8	4	12	12	12	12	12
Key ssssnnnn	Msg. length	Mapping info	Mapping info	Mapping info	Mapping info	Mapping info

Message Header (HD)

8	70
Sequence no.(key)	Text

Message Text (SY)

Figure 4-15. Format of Message Segments

Figure 4-16 shows the layout of the mapping information that defines data fields to be included.

8	3	1
Field name or VARLISTn	Position in the message	Blank space

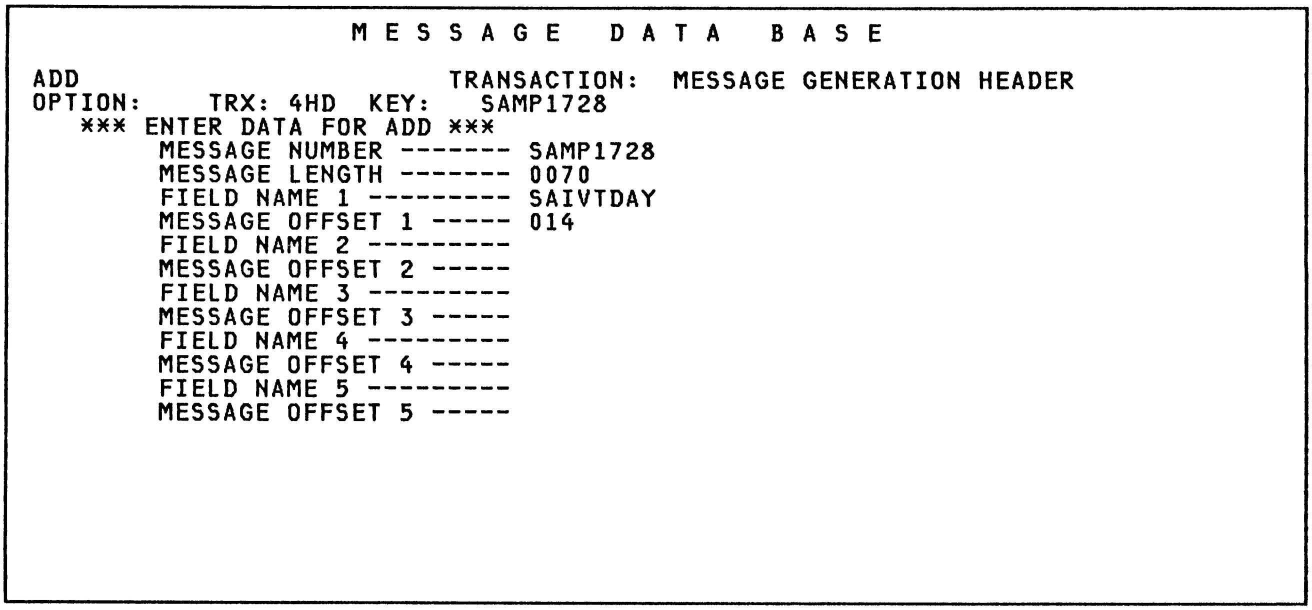
Figure 4-16. Layout of Mapping Information

Space must be allowed when preparing the message text for the values requested. Decimal and packed decimal numbers are edited to allow for a decimal point and a sign; binary numbers also have an edited floating sign. Position numbers commence at 1.

Both field names and VARLIST names can be included in a message. VARLIST names allow system information to be displayed in error messages:

VARLIST1	- DL/I status code
VARLIST2	- Transaction mode and ID (3 characters)
VARLIST3	- User ID
VARLIST4	- Audited field name (8 characters)
VARLIST5	- Value of audited field
VARLIST6	- DB2 status code
VARLIST7	- DB2 warning codes

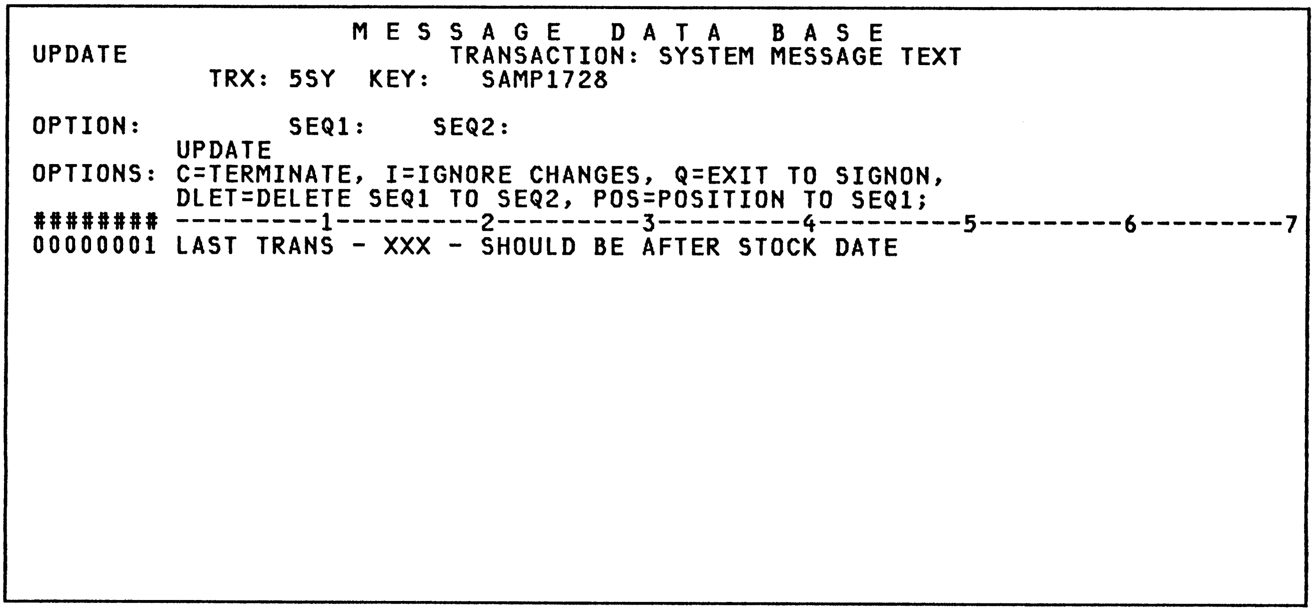
In Figure 4-17, a new message header is created.



```
                     M E S S A G E   D A T A   B A S E

ADD                        TRANSACTION:  MESSAGE GENERATION HEADER
OPTION:    TRX: 4HD  KEY:   SAMP1728
   *** ENTER DATA FOR ADD ***
           MESSAGE NUMBER ------- SAMP1728
           MESSAGE LENGTH ------- 0070
           FIELD NAME 1 --------- SAIVTDAY
           MESSAGE OFFSET 1 ----- 014
           FIELD NAME 2 ---------
           MESSAGE OFFSET 2 -----
           FIELD NAME 3 ---------
           MESSAGE OFFSET 3 -----
           FIELD NAME 4 ---------
           MESSAGE OFFSET 4 -----
           FIELD NAME 5 ---------
           MESSAGE OFFSET 5 -----
```

Figure 4-17. Creating a Message Header

In Figure 4-18, the message text is inserted with allowance for the fields to be mapped in.



```
                     M E S S A G E   D A T A   B A S E
UPDATE                     TRANSACTION: SYSTEM MESSAGE TEXT
               TRX: 5SY  KEY:   SAMP1728

OPTION:           SEQ1:    SEQ2:
            UPDATE
OPTIONS: C=TERMINATE, I=IGNORE CHANGES, Q=EXIT TO SIGNON,
         DLET=DELETE SEQ1 TO SEQ2, POS=POSITION TO SEQ1;
######## ---------1---------2---------3---------4---------5---------6---------7
00000001 LAST TRANS - XXX - SHOULD BE AFTER STOCK DATE
```

Figure 4-18. Inserting Message Text

WARNING MESSAGES

Sometimes it is necessary to warn the user of some unusual but not critical situation, such as a very high discount or a low stock position, but still allow the transaction to complete after the user has had a chance to confirm the intention.

To display a warning message, write:

```
WARNMSG = nnnn
```

in the high level audit language and set up a message of number **nnnn** in the Message Data Base as for error messages.

The user will receive the display just as for error messages, but if only warnings are present, he will be told to enter option **U** to complete the transaction. Alternatively, the user can alter data, and auditing will be done again to verify that the change has not upset another validation requirement. The message leg (P2) will not be used until the U option has been entered when there are warning messages.

A warning message must be associated with a field (as must an error message) and only one such message (warning or error) can be associated with one field. By default a warning message is associated with the audited field (i.e., that named in the preceding FIELD header statement).

To associate a warning message with another field, write:

```
  WARNMSG  SSXXFFFF = nnnn
```

where SSXXFFFF is the field name.

AUTOMATIC FIELD ASSIGNMENT (AFA)

Occasionally, in order to ensure that some audit processing is carried out before the main field audits, it is necessary to place operation descriptors and data descriptors in the left hand leg of the Audit Data Base (see Figure 4-19).

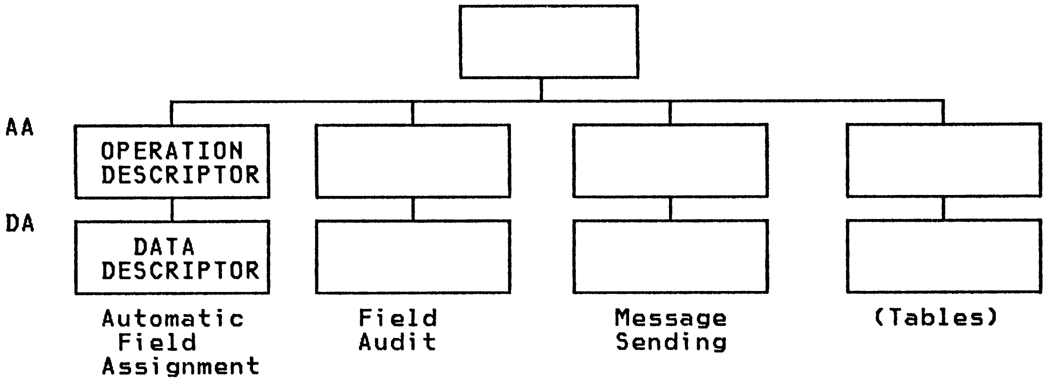


Figure 4-19. AFA in the Audit Data Base

They are coded exactly like field audits, but they are preceded by the header P0 instead of P1. All the AFA rules for all fields in the transaction are executed before any field audits are executed. This fact is sometimes helpful in determining the sequence of audit operations.

You must tell the Auditor to look for AFA rules by coding AFA=YES against the Rules Generator FIELD statements for the fields requiring it. If AFA=YES is coded, rules must be present and they will always be executed, regardless of whether the field has been changed.

AFA rules can raise error messages just as field audits can. If errors are detected during AFA, the Auditor continues to perform the field audits and collects all the error messages together for a single display. Fields in error are redisplayed and highlighted.

Example

When inserting a concatenated segment (a combined view of a logical child and logical parent), it is an IMS/VS requirement that the concatenated key of the logical parent be written in front of the logical child and match the key field in the logical parent. An AFA can be defined to move it from one position to another.

Significant Rules Generator statements:

```
  SEGMENT    ID=CT,...
  FIELD      ID=CKEY,KEY=YES,LENGTH=10,NAME=
  FIELD      ID=PKEY,LENGTH=10,POS=20,AFA=YES
```

High level audit language:

```
  SYSID = SAMP
  SEGID = CT
  FIELD = PKEY
  PROCESS
  P0
  IF MODE = 4
    PKEY = SACTCKEY
  ENDIF
```

Generated segments:

```
  Audit root key (GF):         SAMPYYYYSACTPKEY
  Audit operation desc (AA):   0167        0200     Is this an insertion?
  Audit data desc (DA):          0001(4)
  Audit operation desc (AA):   0210SACTCKEY00       Move CAT key to parent
```

The same effect can be achieved by using field audits and coding AUDIT=YES, FAUDIT=YES on the PKEY field definition.

COMMON AUDITS

If several fields with the same ID in different segments in the same or different application systems have identical auditing requirements (including preaudit, AFA and automatic message sending), they can use common audit rules under a special root segment key in the Audit data base. The key format is:

COMMON000000ffff

where:

COMMON000000 is a literal
ffff is the field ID

To notify the Auditor that a field's audit rules are stored under such a root segment key, code CAUDIT=YES on the Rules Generator FIELD statement.

KEY AUDITING

By coding the KAUDIT operand on the Rules Generator FIELD statement, the services of the Auditor can be requested during primary and secondary key selection (the KEY call) as well as just prior to display (the PRELIM call) and at update time (the PROCESS call).

The principal uses of this capability are:

- Editing keys
- Enforcing key range security
- Editing data on the Secondary Key Selection screen
- Preventing the user from viewing some segment occurrences on the Secondary Key Selection screen

- Transaction switching (see Chapter 6, "Complex Transactions")

The processing to be performed during key audit can be specified in the high level audit language. The primary key audit processing statements are preceded by the header P0 while the secondary key audit statements are preceded by P1. This means that they go into the AFA leg and the field audit leg, respectively, of the Audit Data Base.

Error messages can be produced during key audit by means of the ERRORMSG statement. On an error condition, the Primary Key Selection screen is redisplayed with the keys in error highlighted, and the user is invited to enter **E** to display the error messages. This is the normal way to enforce key range security. High level audit language statements can check the user ID and logical terminal name, and refer to tables or other data bases to complete the checking.

EDITING KEYS

Key fields (fields marked KEY=YES to the Rules Generator) can be edited using key audit. There are two methods that can be used alone or in combination:

- Split the key into subfields, as described in Chapter 2, "Static Rules and the Rules Generator."

  Each subfield is marked KEY=YES to the Rules Generator, but some of them are to be set by audit processing instead of being entered by the user. These are marked KDISP=NO to prevent their being shown to or entered by the user. They will also be marked KAUDIT=PRIM to cause primary key audit to be invoked.

- Where the editing is more than simple insertion or formatting, define a pseudo segment and define one field for each key field to be edited.

  The field in the pseudo segment must be in the displayed format. This pseudo segment field will be the one that the user enters on the Primary Key Selection screen or in the concatenated key area of any of the screens. The pseudo segment field is associated with the key field by naming it as the value of the COFIELD operand on the Rules Generator FIELD statement that defines the key field. Audit logic must be coded to move what the user has entered from the COFIELD into the key field during primary key selection. Additional logic will be needed during secondary key selection to move from the key or related field into the COFIELD in order to let the user see it on the Secondary Key Selection screen and on the Data Display screen.

Example

All the part numbers in the sample system begin with 02. To save the user entering 02 every time, we amend the definition of the root segment.

Significant Rules Generator statements:

```
  SEGMENT ID=PA,PARENT=0,NAME=PARTROOT,LENGTH=50,
          SKSEGS=18,KEYNAME=PARTKEY
  FIELD   ID=PK02,LENGTH=2,KAUDIT=PRIM,KDISP=NO,KEY=YES
  FIELD   ID=KEY,LENGTH=15,KEY=YES,SNAME='PART NUMBER'
  FIELD   ID=DESC,LENGTH=20,POS=27,DISP=YES,REL=YES,
          SNAME='PART DESCRIPTION'
```

High level audit language:

```
  SYSID  =  SAMP
  SEGID  =  PA
  FIELD  =  PK02
  KANAME =  ALT
  KEY
  P0
  PK02 = '02'
```

See "Note on Separation of Calls" on page 4-24 for a discussion of the KANAME assignment. If KANAME = ALT is coded here, it must also be coded on the Rules Generator GENERATE statement that defines the transaction.

CONTROLLING SECONDARY KEY SELECTION

Secondary key audit is requested by writing KAUDIT=SECO on the Rules Generator FIELD statement that defines a key field (KEY=YES) or a related field (REL=YES). The high level audit language statements are headed by KEY and P1. The code that follows these headers is performed once for every segment occurrence until enough segments have been retrieved to fill the secondary key selection screen or until an SKSDISP = STOP statement is encountered.

When the user makes a selection by entering a selection number, that segment is retrieved again; if COFIELD is specified for one or more fields in the segment, the code is performed once more to ensure that the keys will be formatted correctly.

If editing keys or data displayed on the Secondary Key Selection screen is desired, audit processing can be used. A pseudo segment field must be defined for each such field in the data base segment and associated with it by being named in the COFIELD operand of the FIELD statement for the data base field. The data base field must either be a key or a related field.

Selectivity can be introduced on secondary key selection for security or other reasons on the basis of data values. If the high level audit language statement SKSDISP = OFF is executed, the segment occurrence currently being processed is not shown and the next occurrence is retrieved. IMSADF II will continue retrieving until enough unrejected segment occurrences have been retrieved to fill the screen, or until there are none left, or until the high level audit language statement SKSDISP = STOP is encountered. The segment occurrence being processed when the SKSDISP = STOP is encountered will not be displayed.

Example

Following the earlier example of table lookup for country codes, suppose you want to show the country name on the Secondary Key Selection screen and at the same time prevent user ID 123456 from viewing segments with country code DK. The rules would be thus:

Significant Rules Generator statements:

```
  SEGMENT ID=DA,PARENT=...
  FIELD   ID=ABCD,LENGTH=5,KEY=YES
  FIELD   ID=CODE,LENGTH=3,REL=YES,KAUDIT=SECO,COFIELD=CNAM.PS
  SEGMENT ID=PS,TYPE=PS            (Pseudo Segment)
  FIELD   ID=CNAM,LENGTH=20
```

High level audit language:

```
SEGID  = DA
FIELD  = CODE
KANAME = ALT
KEY
P1
IF USERID = '123456'
  IF CODE = 'DK'
    SKSDISP = OFF
  ENDIF
ENDIF
IF DECODED CODE TO SAPSCNAM USING 'CENTRALALLTABLESCOUNTR' OK
  NOP
ENDIF
```

SEQUENCE OF AUDITING

Audit operations are performed call by call, phase by phase. Within each phase, they are carried out in a definite order, according to the layout of the rules defined to the Rules Generator.

Within each segment, fields are audited in the order in which their field statements are written. The order in which segments are audited is determined by the following rules:

- Pseudo segment fields are audited before data base segment fields. The order in which the pseudo segments are audited is determined by the order in which they are named in the TSEGS operand of the GENERATE statement that defines the transaction.

- Auditing for segments in the DBPATH takes place next. Each hierarchical path is audited beginning with the highest level and working down. If a segment is in more than one path, it will be audited only in the first path. If no field from a segment is displayed, the segment will not be loaded - and no auditing can be done on it - unless it is named on the DBPATH operand. The paths are audited in the order in which they are identified in the DBPATH operand.

- Finally, data base segments are processed in the order in which they are named in the TSEGS operand of the GENERATE statement that defines the transaction.

KEY CALL

When key audit is requested for one or more fields, processing during key selection is affected. IMSADF II works down each hierarchical path identified in the DBPATH operand of the GENERATE statement that defines the transaction. It goes through the following steps for each segment in each hierarchical path:

- If the key field is marked KAUDIT=PRIM in the Rules Generator FIELD statement, the Auditor is called to validate or edit the key entered by the terminal user.

- IMSADF II attempts to retrieve the segment from the data base.

- If the retrieval fails and secondary key selection is allowed for that segment (the default for non-root segments), IMSADF II goes through secondary key selection.

- The Auditor is called for each segment occurrence if KAUDIT=SECO is specified for any key or related field in that segment.

IMSADF II then performs the same steps for the next segment.

PRELIM CALL

When preaudit is requested for one or more fields, the Auditor will perform the following steps prior to displaying the segment data in any transaction mode.

- Perform automatic field assignment for fields marked AFA=YES and PAUDIT=YES.
- Perform field audits for fields marked PAUDIT=YES.
- If any errors or warnings have occurred, display an error message screen to the user, logically paged if necessary, and return to the primary key selection screen.
- If there are no errors, execute the message leg of the Audit data base, including automatic message sending, for those fields marked MSG=YES and PAUDIT=YES.

PROCESS CALL

At update time (the PROCESS call) the Auditor will do the following:

- Check fields marked FAUDIT=YES and mark them as changed.
- Check whether required fields (REQ=YES) are non-initialized values and mark them in error if they are. (See the discussion of the REQUIRED operand in the IMS Application Development Facility II Version 2 Release 2 Application Development Reference.)
- Perform automatic field assignment for fields marked AFA=YES.
- For fields marked as changed (e.g., by user input), verify numeric content for TYPE=NUM, verify a valid month and day for TYPE=DATE, and verify alphabetic content for TYPE=ALPHA.
- Perform field audits for fields marked AUDIT=YES and changed.
- If any of steps 2-5 have yielded error or warning messages, redisplay the screen with the fields in error highlighted and the message "ENTER 'E' TO DISPLAY ERROR OR WARNING MESSAGES."
- If user enters E in the OPTION field, show the error and warning messages, logically paging if necessary.
- If only warning messages were raised, allow user to enter option U to continue on to the message leg and complete the transaction.
- When user enters corrections, redo all audit operations (starting at step 2).
- When there are no errors, execute the message leg of the Audit Data Base for those fields marked MSG=YES and changed.

  Note: Step numbers 1-5 will be performed on a field by field basis before proceeding on to step number 6. Any audit operations can be performed here. The function is not restricted to automatic message sending. For example, calculations, table manipulation, and DL/I calls are allowed. However, no error error messages can be sent.

NOTE ON SEPARATION OF CALLS

Internally, the compiler keeps the coding for the three calls of auditing separate by means of a coding convention. Unless this convention is understood, unexpected results can sometimes occur.

Audit operation code 16 distinguishes between preaudit (PRELIM) and update (PROCESS). When the compiler encounters the heading PRELIM, it places an operation descriptor 16 on the Audit Data Base. This causes the Auditor to branch to the PRELIM code at preaudit time and to the PROCESS code at update time. However, if no PRELIM call is coded in the high level audit language, the compiler does not generate an operation descriptor with code 16. If PAUDIT=YES is nevertheless coded on the Rules Generator FIELD statement, the Auditor will not be able to distinguish between PRELIM code and PROCESS code and will perform all of the PROCESS code both at preaudit time and at update time. This consideration applies separately on each of the three legs of the data base (AFA - P0, Field Audit - P1, and Message - P2).

Therefore, whenever a field is marked PAUDIT=YES, it is wise to code all three phases (P0, P1, P2) in the PRELIM call.

For example:

```
  SYSID = SAMP
  SEGID = PA
  FIELD = DESC
  PRELIM
  P0
  NOP
  P1
  * Actual code here
  P2
  NOP
```

A similar convention is employed to separate key audit processing. In this case audit operation descriptor F2 distinguishes between key audit and the other calls. This convention is observed by default only when KANAME = ALT is coded on the DEFADF macro (see the IMS Application Development Facility II Version 2 Release 2 Installation Guide). If this is not done, it must be requested by coding KANAME = ALT before the processing logic, as shown in earlier examples of key auditing. KANAME=ALT must also be coded on the Rules Generator GENERATE statement that defines the transaction.

If these keywords are not coded, another convention for separation is used, whereby all key audit logic is stored under a key of form:

KEYAUDITSSXXFFFF

where:

KEYAUDIT is a literal
SSXXFFFF is the name of the audited field

SUMMARY OF RULES GENERATOR OPERANDS FOR AUDITING

AUDIT=YES Executes field audits during Process call on fields marked as changed.

FAUDIT=YES Forces an audit by marking field as changed (assumed for key fields of segments retrieved via key selection).

REQ=YES Field must have non-initialized values.

MSG=YES Uses message sending logic if field changed during update and during preaudit if PAUDIT=YES.

AFA=YES Requests phase P0 during update call and during preaudit if PAUDIT=YES.

PAUDIT=YES Causes field audit rules to be executed during preaudit (prior to data display).

CAUDIT=YES Directs the Auditor to find the rules under the root key COMMON000000ffff where ffff is the field ID.

ASTATUS=YFRMAPC Equivalent to all of the above (in the same order). Select from the seven possible values but do not mix ASTATUS with the alternative individual operands.

KAUDIT=PRIM Requests primary key audit.

KAUDIT=SECO	Requests secondary key audit.
KAUDIT=YES	Requests both primary and secondary key audit.
KDISP=NO	Prevents the user from seeing or entering this key field.

USER-WRITTEN AUDIT ROUTINES

Certain audit operation codes are reserved to allow you to produce your own operations by writing audit exits in COBOL, PL/I or Assembler. These operation codes are:

```
    70 to 99 and
    W0 to Z9
         (70 total)
```

The AEXIT statement in the high level audit language must be used with these codes.

When the Auditor encounters operation descriptors bearing one of these codes, it branches to the exit routine, which performs processing, accessing and assigning fields, and DL/I calls as desired. The exit routine returns a true/false indicator to the Auditor to enable it to continue operation in the normal way. Chapter 9, "Exits" explains how to implement exits.

BATCH INPUT OF DYNAMIC RULES

Dynamic rules can be entered in bulk in the Audit and Message data bases. For the high level audit language, use the supplied JCL procedure ????AL.

Note: **????** is the installed ADFID (the default is MFC1).

For example:

```
  //  EXEC ????AL
  //INDATA DD *
  SYSID = SAMP
  SEGID = ..
  etc.
  /*
```

For entering error messages, batch processing is usually more convenient when large numbers of messages must be coded. It is then helpful to use the online facilities to amend rules during testing and maintenance. If batch input is kept in step with online changes, it can be used again to enter the rules into a production system when tests are complete. The production system can be implemented as a different application system ID in the same facility libraries and data bases or in separate facility libraries and data bases under a separate IMS/VS control region.

Here is an example of batched input for an error message:

```
  //        EXEC   ????B
  //TRANSIN DD   *
  *NOW THE ERROR MESSAGE TEXT
  MFC1B3HD
  SAMP9400
  MFC1B4HD
  SAMP94000070SAPDPLRV034 SAPDCOMM061
  MFC1B4SYSAMP9400
  00000001WARNING: PLANNED REVISION NUMBER(   ) IS LESS THAN COMM CODE(   )
  /*
```

Note that whenever it is necessary to insert a root segment, it is deleted beforehand. In this way the deck can be re-run as often as necessary. On the first run there will be error messages, since the segments will not be found. These can be ignored.

As can be seen, each batch transaction begins with transaction code:

ssssBmtx

where:

ssss is the application system ID (in this case MFC1)
B is a literal
m is the transaction mode (1 to 5)
tx is the transaction ID

Input records must be 80-byte card images, with transaction code in column 1.

BATCH INPUT LAYOUTS - AUDIT DATA BASE (TABLES)

The following tables show the batch input layout for Audit Data Base tables.

TN - Table Name

Card	Column	Length	Description
1	9	16	Key of GF segment
	25	6	Key of TN segment
	31	22	Table description

Sample:

```
MFC1B4TNSAMPYYYYSACDTABLTABLE1THIS IS TABLE#1
```

TA - Table Entry

Card	Column	Length	Description
1	9	16	Key of GF segment
	25	6	Key of TN segment
	31	8	Key of TA segment (argument)
2	1	70	Table value

Sample:

```
MFC1B4TASAMPYYYYSACDTABLTABLE11234
```

Note: This is the value for argument 1234.

BATCH INPUT LAYOUTS - MESSAGE DATA BASE

The following tables show the layout for batch input for an error message.

HD - Message Generation Header

Card 1 has the HD transaction name MFC1B4HD.

Card	Column	Length	Description
2	1	8	Application system ID and message number
	9	4	Message length
	13	8	Field name to be mapped from
	21	3	Offset in text where data is to be mapped
	25	8	Mapping information for 1 to 5 data mappings
	33	3	"
	37	8	"
	45	3	"
	37	8	"
	57	3	"
	61	8	"
	69	3	"

Sample:

```
MFC1B4HD
SAMP99990070SACDDIPU030
```

SY - Message text

Card	Column	Length	Description
1	9	8	Key of HD segment
2	1	8	Sequence number of SY segment
	9	70	Message text

Sample:

```
MFC1B4SYSAMP9999
00000001DISBURSEMENT CODE INCORRECT (-) SPECIFY P OR U
```

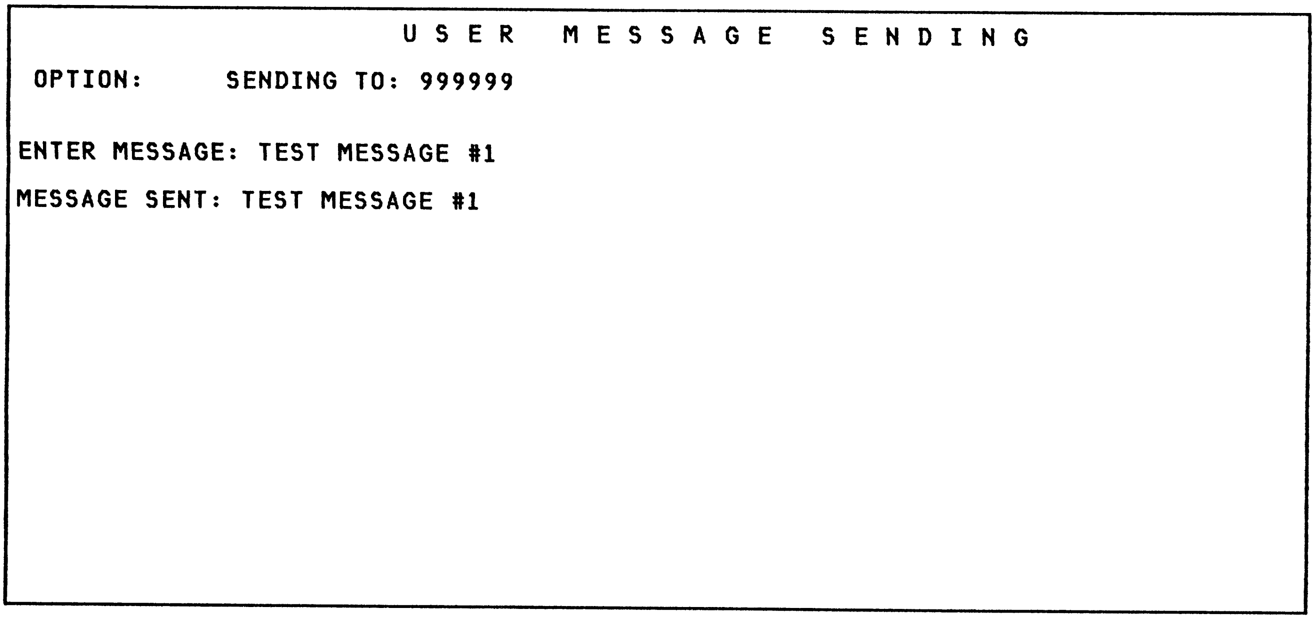
CHAPTER 5. MESSAGE SENDING AND DISPLAY

The IMSADF II Dynamic Rules Data Bases can be in one of two formats. They can be either DL/I hierarchical data bases or DB2 relational data bases. This chapter describes the DL/I form of the data bases. For additional information about the DB2 form of the IMSADF II Dynamic Rules Data Bases, see Chapter 4 of the IMS Application Development Facility II Version 2 Release 2 Application Development Reference.

Conversational applications allow the user to select options A, B, H and I if they are included in the POMENU operand of the SYSTEM statement. They are:

A - project message sending
B - project message display
H - user message sending
I - user message display

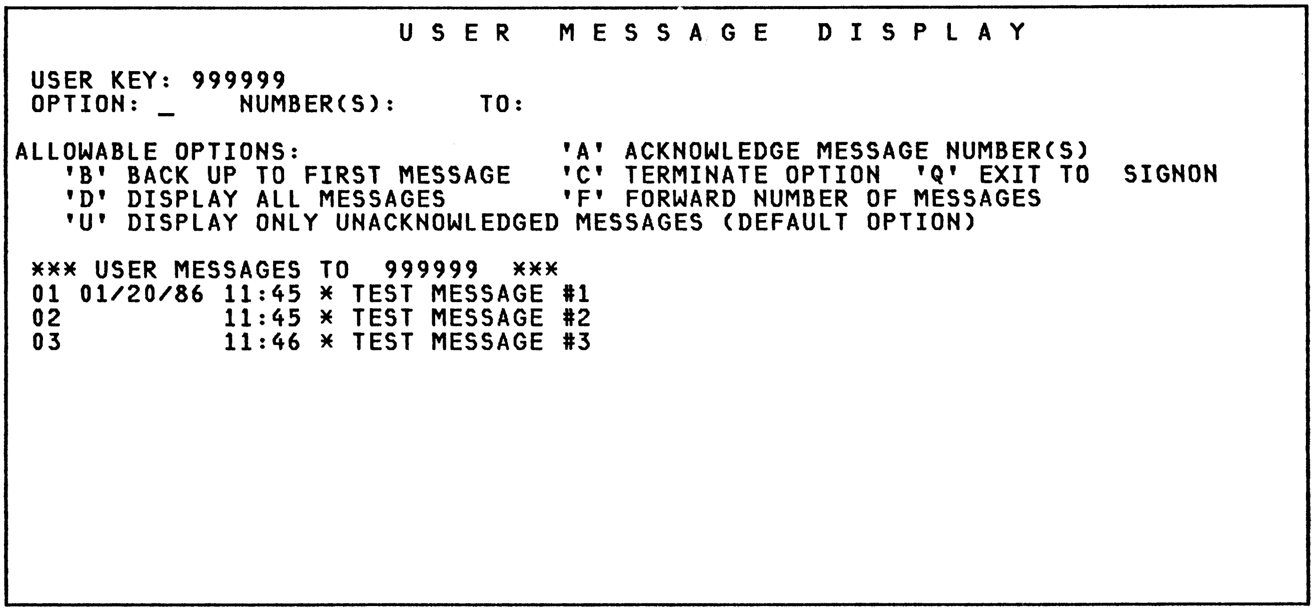
The functions are illustrated in the following figures. These are information messages which are not sent directly to a terminal but are collected in data bases to be viewed by the B and I options.



```
                        USER  MESSAGE  SENDING
 OPTION:     SENDING TO: 999999

ENTER MESSAGE: TEST MESSAGE #1

MESSAGE SENT: TEST MESSAGE #1
```

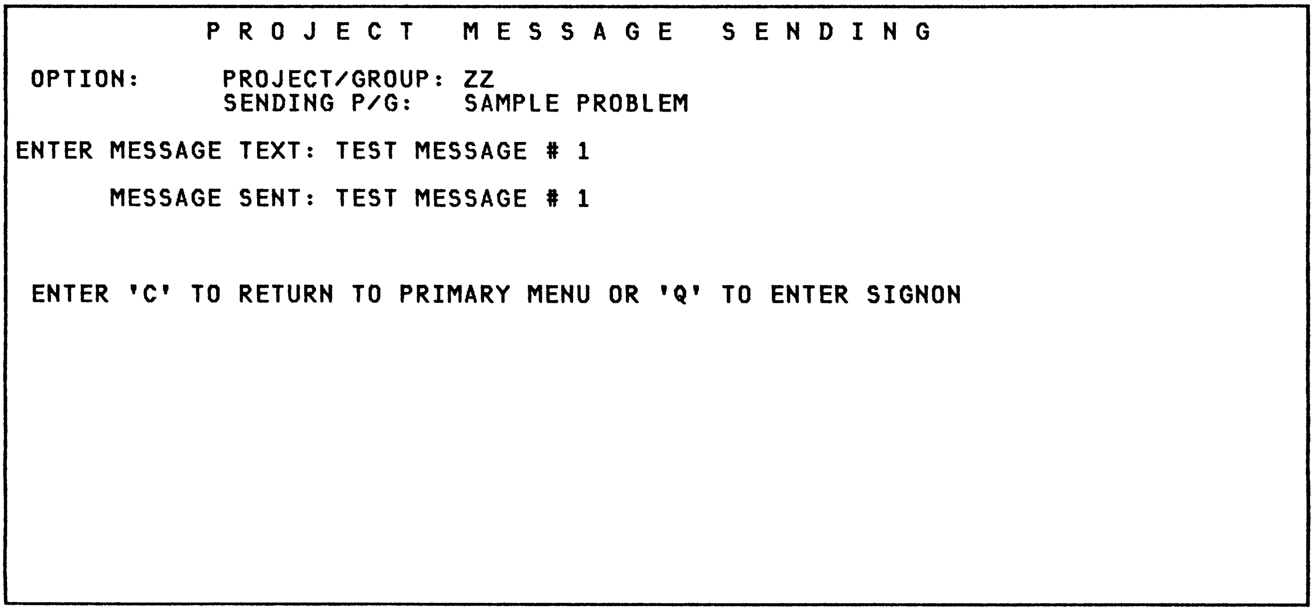
Figure 5-1. User Message Sending

As can be seen in Figure 5-2 and Figure 5-4, the user can acknowledge messages, which are then removed from the display. However, they are still present on the data base and can be seen by means of OPTION D (DISPLAY ALL MESSAGES).



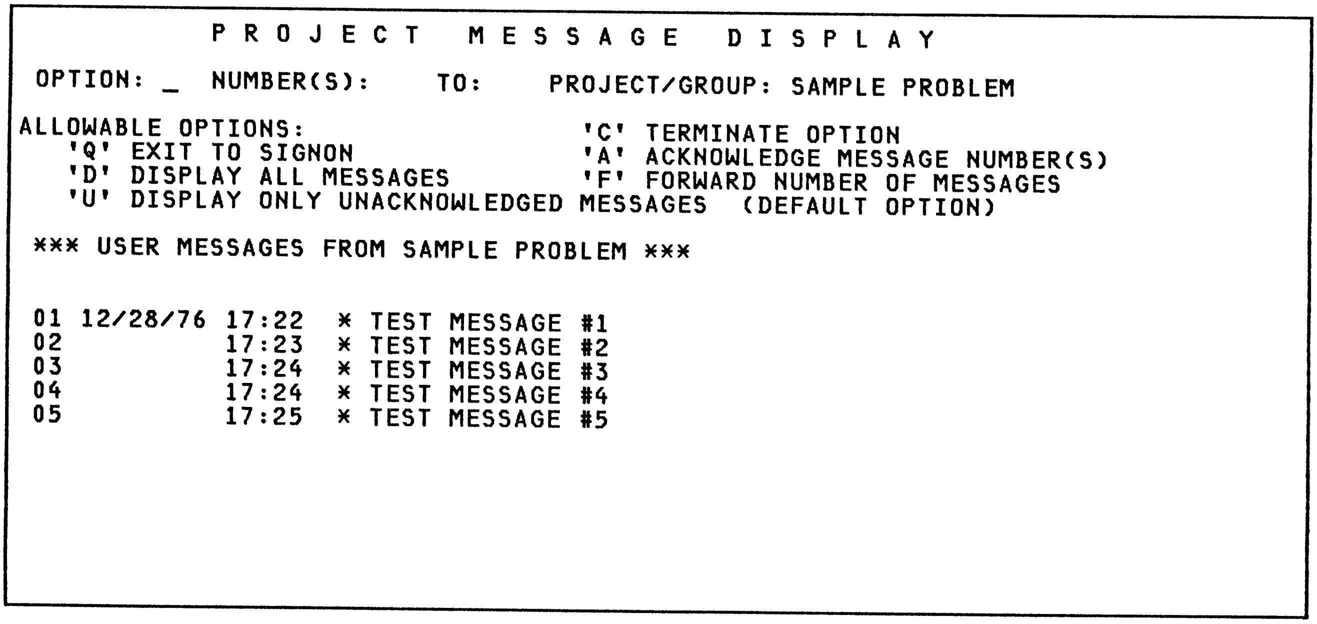
```
                        U S E R   M E S S A G E   D I S P L A Y

 USER KEY: 999999
 OPTION: _    NUMBER(S):     TO:

ALLOWABLE OPTIONS:                      'A' ACKNOWLEDGE MESSAGE NUMBER(S)
   'B' BACK UP TO FIRST MESSAGE   'C' TERMINATE OPTION  'Q' EXIT TO  SIGNON
   'D' DISPLAY ALL MESSAGES       'F' FORWARD NUMBER OF MESSAGES
   'U' DISPLAY ONLY UNACKNOWLEDGED MESSAGES (DEFAULT OPTION)

 *** USER MESSAGES TO  999999  ***
 01 01/20/86 11:45 * TEST MESSAGE #1
 02          11:45 * TEST MESSAGE #2
 03          11:46 * TEST MESSAGE #3
```

Figure 5-2. User Message Display



```
            P R O J E C T   M E S S A G E   S E N D I N G

 OPTION:     PROJECT/GROUP: ZZ
             SENDING P/G:   SAMPLE PROBLEM

ENTER MESSAGE TEXT: TEST MESSAGE # 1

      MESSAGE SENT: TEST MESSAGE # 1


 ENTER 'C' TO RETURN TO PRIMARY MENU OR 'Q' TO ENTER SIGNON
```

Figure 5-3. Project Message Sending



```
             P R O J E C T   M E S S A G E   D I S P L A Y
 OPTION: _  NUMBER(S):    TO:    PROJECT/GROUP: SAMPLE PROBLEM
ALLOWABLE OPTIONS:                     'C' TERMINATE OPTION
   'Q' EXIT TO SIGNON                  'A' ACKNOWLEDGE MESSAGE NUMBER(S)
   'D' DISPLAY ALL MESSAGES            'F' FORWARD NUMBER OF MESSAGES
   'U' DISPLAY ONLY UNACKNOWLEDGED MESSAGES  (DEFAULT OPTION)
 *** USER MESSAGES FROM SAMPLE PROBLEM ***

 01 12/28/76 17:22  * TEST MESSAGE #1
 02          17:23  * TEST MESSAGE #2
 03          17:24  * TEST MESSAGE #3
 04          17:24  * TEST MESSAGE #4
 05          17:25  * TEST MESSAGE #5
```

Figure 5-4. Project Message Display

IMSADF II uses its own data bases to hold the information messages. Messages to project/groups go into the Sign-On Profile Data Base, since it has root segments keyed on project/group. Segment usage is shown in Figure 5-5.

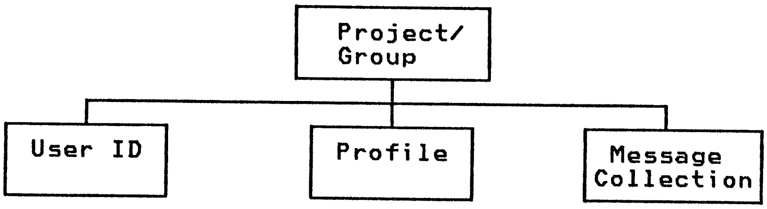


Figure 5-5. Project Message Collection in the Sign-On Profile Data Base

Messages to users go into the Message Data Base in a separate segment type (see Figure 5-6). Before messages can be sent to a user in this way, a header must be created; transaction UH (User Header) is provided for the purpose in the MFC1 application system.

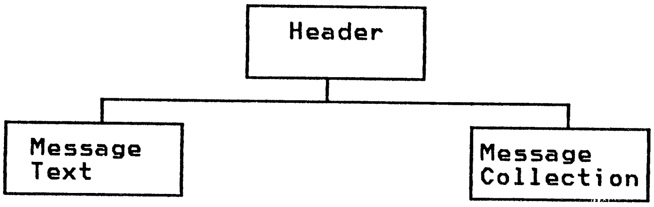


Figure 5-6. User Message Collection in the Message Data Base

MESSAGE MAINTENANCE

A batch utility is supplied to print messages and delete those that have been acknowledged. An example of JCL to execute Message Maintenance as an IMS/VS batch job is shown below. Similar JCL using the IMSBATCH procedure may be used to execute Message Maintenance as a BMP transaction.

To run under CICS/OS/VS, the same DD statements must be added to the CICS/OS/VS Startup Job Stream. For additional information on running the IMSADF II batch driver under CICS/OS/VS, see the _IMS Application Development Facility II Version 2 Release 2 Application Development Reference_.

```
//MAINT     JOB   ACCNT,NAME,MSGLEVEL=1
//MAINT     EXEC  DLIBATCH,MBR=????BDCT
//MSGOUT    DD    SYSOUT=A
//PRINTER   DD    SYSOUT=A
//RSTRTIN   DD    DUMMY
//TRANSIN   DD    *

MFC1B3MM PG     ZZ     YY     XX     WW     VV
MFC1B3MM PG     UU
MFC1B3MM USER   999999 888888
/*
```

Note:
For **????**, substitute the ADFID of your installed system.

Note card columns:

```
1        10     16     23     30     37     44
```

The example will print and delete messages for project/groups ZZ, YY, XX, WW, VV, and UU and for user IDs 999999 and 888888. Instead of coding individual requests, the following options may be used starting in column 10 with no following codes:

```
ALLPG - all project/groups
ALLUS - all user IDs
ALL   - all user IDs and project/groups
```

Since the messages are on data bases, they are also accessible to application programs and to ordinary IMSADF II transactions.

AUTOMATIC MESSAGE SENDING

In addition to having user and project messages entered by a terminal user employing options A or H, they can be triggered automatically during IMSADF II transactions. The messages are stored on the same data bases and can be acknowledged and maintained in the same manner as user- or project-sent messages. The format is slightly different: automatic messages are date- and time-stamped.

Message sending is under the control of the Auditor, and it uses a different pair of operation and data descriptor segments. These are formatted like field audits, except that the message numbers have a different meaning. Figure 5-7 shows the message leg of the Audit Data Base. Using the high level audit language compiler, : these message leg rules are coded just like other audit logic, except that they are headed P2 instead of P1 or P0. Online, they are maintained by transactions MA and DM, which look exactly like their counterparts FA and DF.

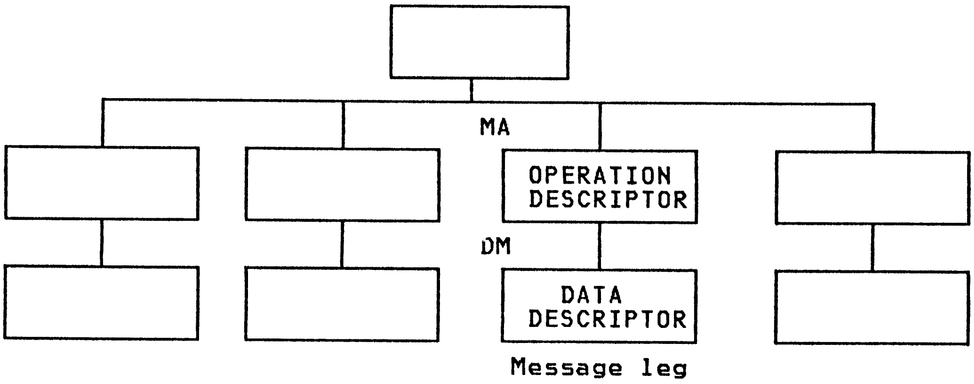


Figure 5-7. Audit Data Base -- Message Leg

Whereas the message number provided sufficient information to send an error message to the online terminal in field auditing, more information must now be specified as to the destination of the messages. Therefore, an extra link is introduced in the chain for the purpose of providing routing information. Thus, an error message number (defined in the ERRORMSG = statement of the high level audit language) linking FA directly to HD, the message header, is replaced by a routing code or information message number (defined in the INFOMSG = statement of the high level audit language) which identifies a routing header (AH). (See Figure 5-8.) AH and AR are simply the transactions that allow creation and maintenance of the routing information. (In fact, AH and AR use the same segment types in the Message Data Base as HD and SY. They rely on a key format convention to keep them apart.)

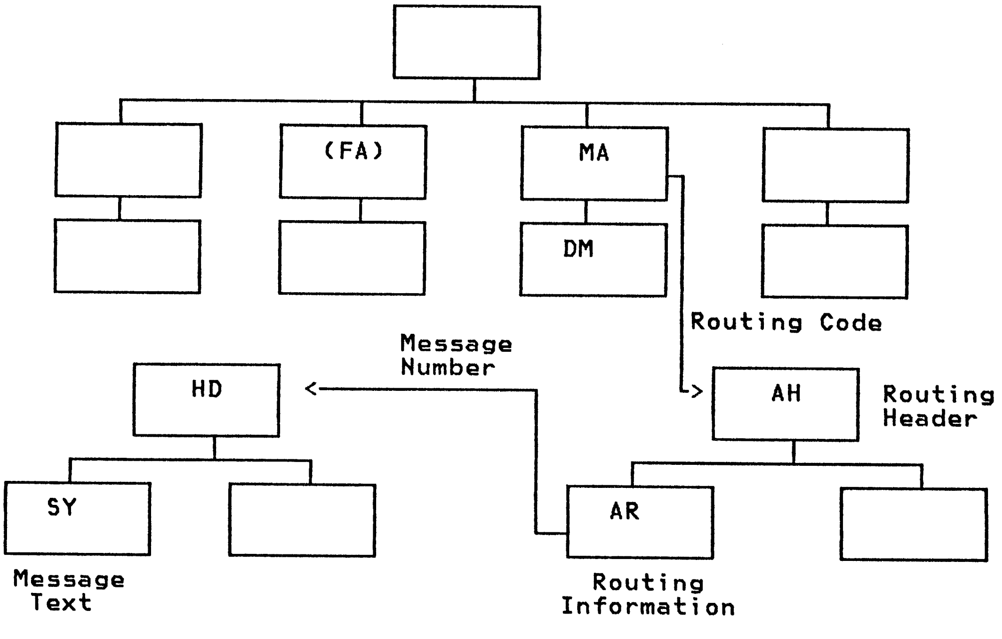


Figure 5-8. How Message Routing Information Fits into the Picture

The numbering and layout of messages (created with HD and SY) are the same as for error messages, except that messages to user IDs are restricted to 127 bytes. The message number, however, is now part of the routing information, rather than being quoted in the audit operation descriptor. The operation descriptor links to the routing header through a routing code. Each routing header (AH) has a routing code that consists of a two-digit number (01 to 99) that is unique within an application system. The routing header key is of the form:

ss####nn

where:

ss is the first two characters of the application system ID
is a literal
nn is the routing code

Beneath the routing header, which has no data, one or more routing information segments are inserted. As Figure 5-9 shows, up to five message numbers can be present. These messages will be sent to the project/group or user ID coded (or to both if both are coded). The sequence number will normally be 00000001. If several segments are present, all the specified messages will be sent as coded.

8	2	4	4	4	4	4	6
Sequence number (key)	Project/ Group	Msg Number	Msg Number	Msg Number	Msg Number	Msg Number	User ID

Figure 5-9. Format of Message Routing Information

In the audit operation descriptors that request automatic message sending, a two-digit routing code must be defined instead of a four-digit message number. A four-digit area is provided for consistency of layout, however, and has the form:

0fnn

where:

0 is a literal
f is a format code which normally has the value 3
nn is the routing code, which must match a corresponding routing header

Format codes are explained in the next section.

The Auditor will execute operations coded in the message leg only if all the following conditions are satisfied:

- All audits in the first and second legs for all fields have completed without error (with user correction if necessary).
- Any field requiring message leg operations is marked MSG=YES on the Rules Generator FIELD statement.
- The fields have been changed or are key fields (KEY=Y). The Auditor performs message leg operations only on key fields or on fields that have been changed (or marked as changed) by previous audits or by FAUDIT=YES on the Rules Generator FIELD statement.

Example

When field MKDP in segment PD in the SAMP system is updated in the range (1200,1400), send message 9375 to project/group XX.

Significant Rules Generator statements:

```
SEGMENT ID=PD,...
FIELD   ID=MKDP,LENGTH=4,MSG=YES
```

High level audit language:

```
SYSID = SAMP
AGROUP = YYYY
SEGID = PD
FIELD = PLRV
PROCESS
P2
* AUTOMATIC MESSAGE SENDING
IF PLRV IN 1200 : 1400
  INFOMSG = 0345
  ENDIF
```

```
Message routing header (AH):  SA####45
Message routing info (AR):    00000001 XX 9375
```

FORMAT CODES IN AUTOMATIC MESSAGE SENDING

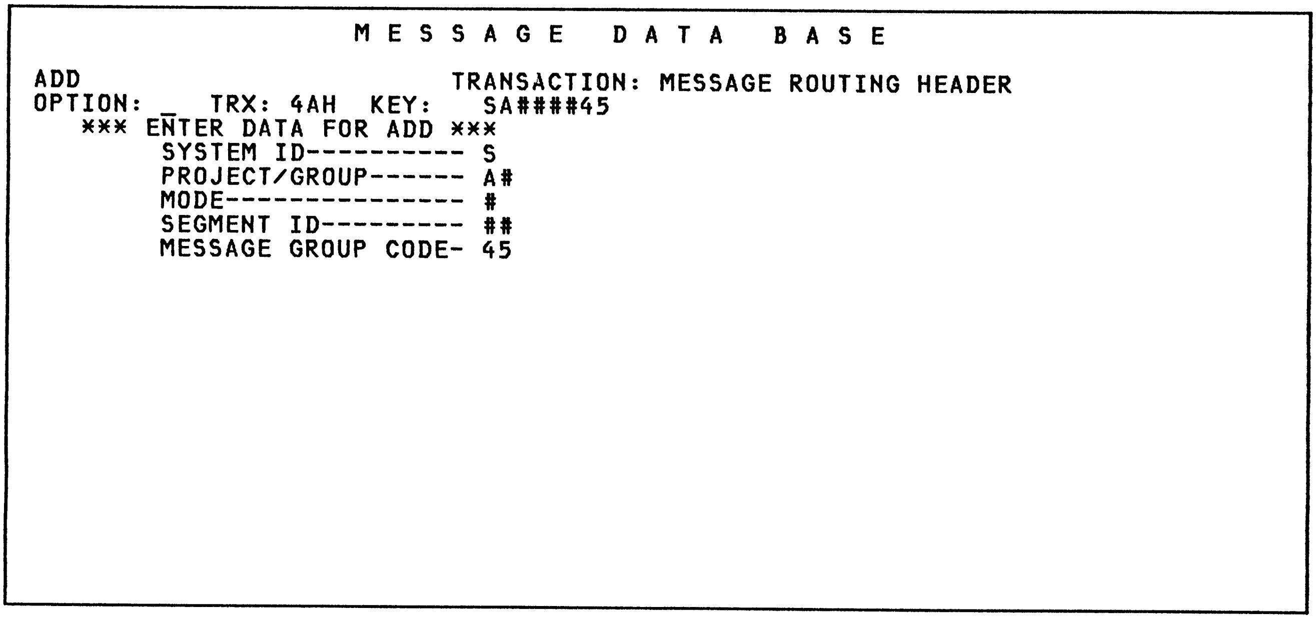
The above example has the information message (INFOMSG) number, which is of the form **0fnn**, using format code f=3, which is the recommended usage. The Auditor will be prompted to locate a message routing header with format ss####nn. There are, in fact, four allowable formats for message routing headers, each selected by a different format code:

Format Code	**Format of AH Key**
f=0	spgmtxnn
f=1	ss######
f=2	ss##pg##
f=3	ss####nn

where:

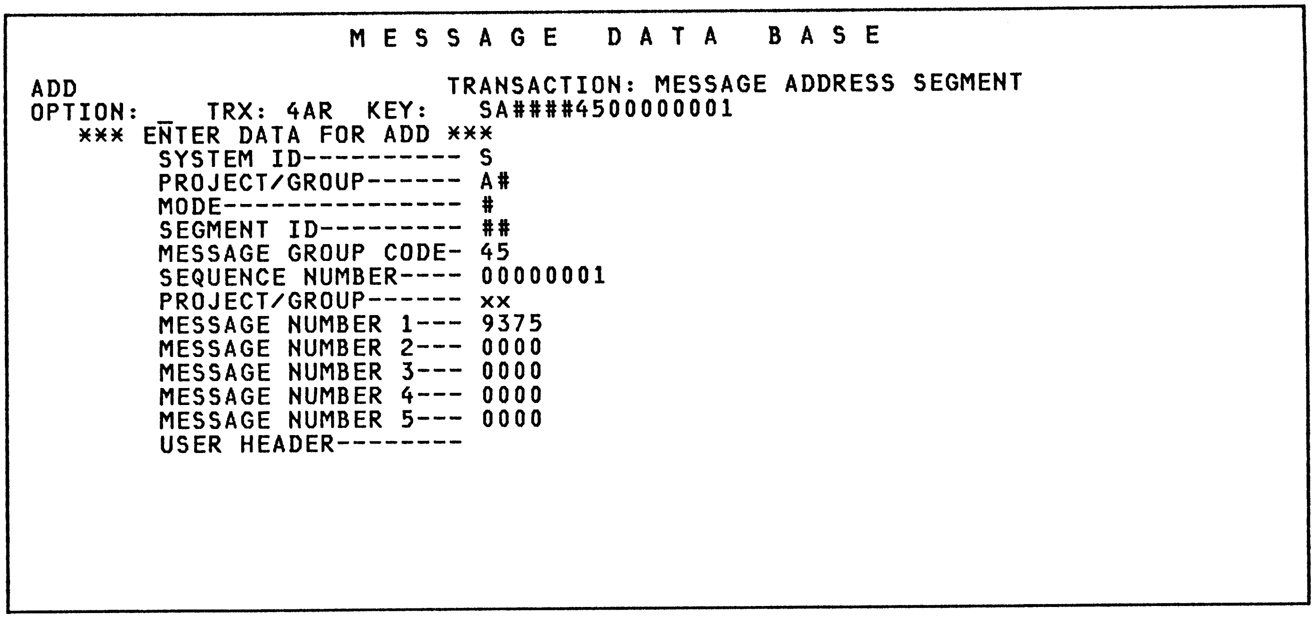
ss is the first two characters of the application system ID
s is the first character of application system ID
nn is the message routing code
pg is the project/group currently signed on
mtx is the current transaction mode (1-6) and ID
is a literal

The screens to enter and maintain message routing information are shown in Figure 5-10 and Figure 5-11. They are designed for format code f=0 for historical reasons.



```
                         M E S S A G E   D A T A   B A S E

 ADD                           TRANSACTION: MESSAGE ROUTING HEADER
 OPTION: _  TRX: 4AH  KEY:   SA####45
    *** ENTER DATA FOR ADD ***
         SYSTEM ID---------- S
         PROJECT/GROUP------ A#
         MODE--------------- #
         SEGMENT ID--------- ##
         MESSAGE GROUP CODE- 45
```

Figure 5-10. Defining a Message Routing Header



```
                      M E S S A G E   D A T A   B A S E

 ADD                        TRANSACTION: MESSAGE ADDRESS SEGMENT
 OPTION: _   TRX: 4AR  KEY:   SA####4500000001
    *** ENTER DATA FOR ADD ***
         SYSTEM ID---------- S
         PROJECT/GROUP------ A#
         MODE--------------- #
         SEGMENT ID--------- ##
         MESSAGE GROUP CODE- 45
         SEQUENCE NUMBER---- 00000001
         PROJECT/GROUP------ xx
         MESSAGE NUMBER 1--- 9375
         MESSAGE NUMBER 2--- 0000
         MESSAGE NUMBER 3--- 0000
         MESSAGE NUMBER 4--- 0000
         MESSAGE NUMBER 5--- 0000
         USER HEADER--------
```

Figure 5-11. Adding Message Routing Information

UNCONDITIONAL AUTOMATIC MESSAGE SENDING

In the case of transactions that work on a single hierarchical path in a data base, the target segment can be deleted by the user selecting transaction mode 3 (or 1). If the user does not modify fields, but merely presses ENTER to delete the segment, the Auditor will not have any fields to audit unless some are marked FAUDIT on the Rules Generator FIELD statement. But there may still be a message sending requirement associated with the transaction rather than with a particular field.

To allow for this case, unconditional message sending is provided. Routing information and message text must still be prepared but, instead of writing audit rules for this case, simply code DAMSG=YES on the transaction GENERATE statement. (DAMSG stands for Delete-Add Message). The routing header key must then be defined in the format:

spgmtx00

where:

s	is the first character of the application system ID
pg	is the project/group currently signed on
mtx	is the current transaction mode (3 or 1 for delete) and ID
00	is a literal

Example

When someone in project/group ZZ deletes (mode 3) a PA root segment using the PA transaction in the SAMP system, send message number 9428 to project/group XX.

Significant Rules Generator statement:

```
GENERATE     TRXID=PA,TRXNAME='PART ROOT',
             OPT=CVALL,PGROUP=ZZ,DAMSG=YES
```

```
Message routing header (AH):  SZZ3PA00
Message routing info (AR):    00000001 XX 9428
```

When segments are being added (transaction mode 4 or 2), unconditional message sending can also be invoked by coding DAMSG=YES on the transaction GENERATE statement and setting up a message routing header with m=4 or 2 (e.g., SZZ4PA00). The message will be sent only during add, however, if at least one field in the transaction is marked MSG=YES on the Rules Generator FIELD statement and if at least one field is changed (or marked as changed) during execution of the transaction. No rules need be coded in the message leg of the Audit Data Base to achieve unconditional message sending.

BATCH INPUT OF DYNAMIC RULES

Dynamic rules for automatic message sending can be entered in bulk using batch input. You may find this method of entry more convenient when large numbers of rules must be coded. Online facilities can then be used to amend rules during testing and maintenance.

Below is an example of IMS/VS batch for automatic message sending.

To run under CICS/OS/VS, the same DD statements must be added to the CICS/OS/VS Startup Job Stream.

```
//RULES     JOB   ACCNT,NAME,MSGLEVEL=1
//          EXEC  DLIBATCH,MBR=????BDCT
//MSGOUT    DD    SYSOUT=A
//PRINTER   DD    SYSOUT=A
//RSTRTIN   DD    DUMMY
//TRANSIN   DD    *
*THE ERROR MESSAGE TEXT
MFC1B3HD
SAMP9400
MFC1B4HD
SAMP94000070SAPDPLRV034 SAPDCOMM061
MFC1B4SYSAMP9400
00000001WARNING:  PLANNED REVISION NUMBER(   ) IS LESS THAN COMM CODE(   )
/*
```

Note: ???? is the installed ADFID (the default is MFC1).

Note that whenever it is necessary to insert a root segment, it is deleted beforehand. In this way, the deck can be rerun as often as necessary. On the first run there will be error messages, since the segments will not be found. These can be ignored.

Each batch transaction begins with a transaction code

ssssBmtx

where:

ssss is the application system ID (in this case MFC1)
B is a literal
m is the transaction mode (1 to 5)
tx is the transaction ID

Input records must be 80-byte card images, with the transaction code in column 1.

BATCH INPUT LAYOUTS

AH - Automatic Message Sending Header

Card	Column	Length	Description
1	9	8	Conditions for automatic message sending

Sample:

```
MFC1B2AHM004IY05
```

AR - Auto Message Routing

Card	Column	Length	Description
1	9	8	Key of AH segment
	17	8	Sequence number of AR segment
	25	2	Project/group to receive message
	27	4	Message number
	31	4	Message number
	35	4	Message number
	39	4	Message number
	43	4	Message number
	47	8	6-character user ID

Sample:

```
MFC1B4AR00000001MX0901                999999
```

HD - Message Generation Header

Card 1 has the HD transaction name MFC1B2HD.

Card	Column	Length	Description
2	1	8	Application system ID and message number
	9	4	Message length
	13	8	Field name to be mapped from
	21	3	Offset in text where data is to be mapped
	25	8	-
	33	3	-
	37	8	- (1 to 5 data mappings)
	45	3	-
	49	8	-
	57	3	-
	61	8	-
	69	3	-

Sample:

```
MFC1B2HD
SAMP99990070SACDDIPU030
```

SY - Message Text

Card	Column	Length	Description
1	9	8	Key of HD segment
2	1	8	Sequence number of SY segment
	9	70	Message text

Sample:

```
MFC1B4SYSAMP9999
00000001DISBURSEMENT CODE INCORRECT (-) SPECIFY P OR U
```

SD - Secondary Transaction Destination

Card	Column	Length	Description
1	9	8	Output Format Rule name (key)
	17	8	Default destination
	25	62	Comments for user information
2	1	71	Comments continued

Use the end of message characters to indicate end of data if a second card is not used. The end of message characters are defined at installation time (DEFADF). The default is $$.

Sample:

```
MFC1B2SDMFORPD01IOPCB   LABOR ERRORS SECONDARY XACT $$
```

LT - Logical Terminal Segment

Card	Column	Length	Description
1	9	8	Key of SD segment
	17	8	Entering LTERM (key)
	25	8	Receiving LTERM #1
	33	8	Receiving LTERM #2
	41	8	Receiving LTERM #3
	49	8	Receiving LTERM #4
	57	8	Receiving LTERM #5
	65	8	Receiving LTERM #6
	73	8	Receiving LTERM #7
2	1	8	Receiving LTERM #8

Use the end of message characters to indicate end of data if a second card is not used. The end of message characters are defined at installation time (DEFADF). The default is $$.

Sample:

```
MFC1B4LTMFORPD01L3277099L3286001IOPCB $$
```

UH - User Header Segment

Card	Column	Length	Description
1	9	8	User ID (Key)
	17	22	User's name

Sample:

```
MFC1B2UH999999  J.SMITH
```

CHAPTER 6. COMPLEX TRANSACTIONS

Chapter 2, "Static Rules and the Rules Generator" described transactions that display and update multiple hierarchical paths in multiple data bases. This chapter presents more advanced application functions and deals with arbitrary combinations of inserting, deleting, and replacing segments.

Great use can be made of the ability to issue DL/I calls using audit operations. In addition to giving control over complex updating, it provides:

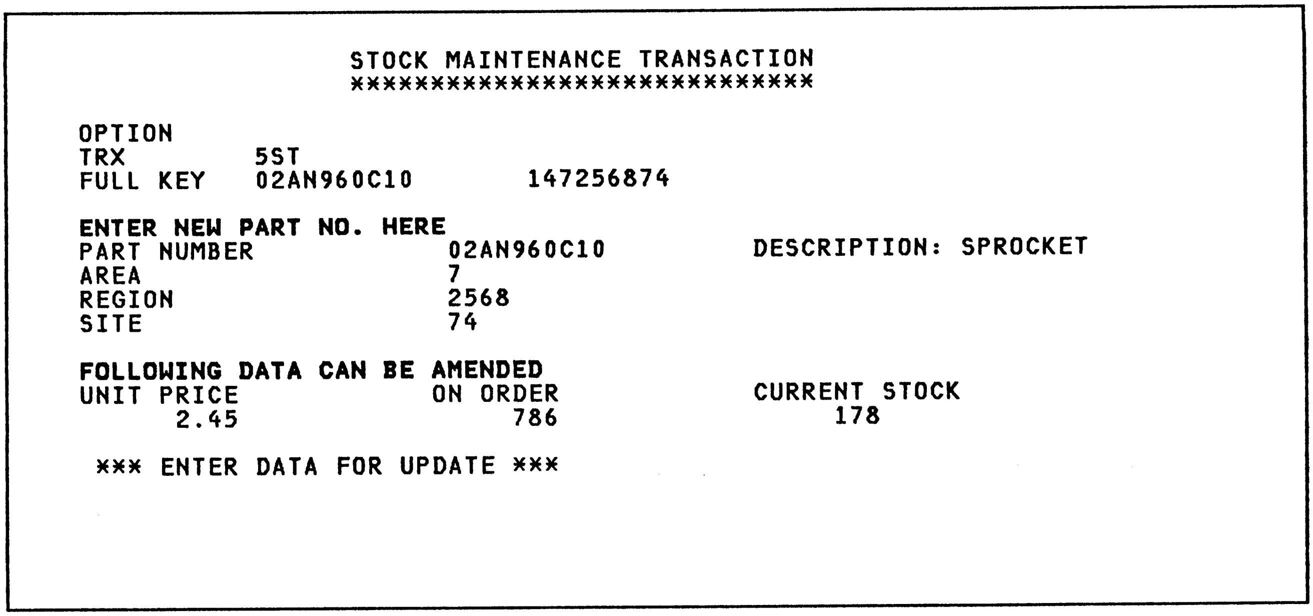
- data validation capability
- one method of processing twins (multiple segment occurrences)
- transaction switching under control of the Auditor
- tailoring the layouts of the Segment Display and Sign-on screens

TAILORING THE DATA DISPLAY SCREEN

Figure 6-1 shows the Rules Generator statement necessary to produce the screen shown in Figure 6-2.

```
         GENERATE  TRXID=ST,TRXNAME='STOCK MAINT',DBPATH=IV,
         OPT=CVALL,SPOS=SIMAGE
&=1
                     STOCK MAINTENANCE TRANSACTION
                     *****************************
&=1
    OPTION    &OPTION
    TRX       &TRAN
    FULL KEY  &KEY
&=1
   'ENTER NEW PART NO. HERE
    PART NUMBER           &5KEY.PA             DESCRIPTION:&6DESC
    AREA                  &5AREA
    REGION                &5REGN
    SITE                  &5SITE
&=1
   'FOLLOWING DATA CAN BE AMENDED
    UNIT PRICE            ON ORDER             CURRENT STOCK
    &5PRIC                &5ONOR               &5STCK
&=1
    &SYSMSG
&ENDS
```

Figure 6-1. GENERATE Statement for Tailored Data Display Screen



```
                    STOCK MAINTENANCE TRANSACTION
                    *****************************

    OPTION
    TRX       5ST
    FULL KEY  02AN960C10       147256874

    ENTER NEW PART NO. HERE
    PART NUMBER             02AN960C10          DESCRIPTION: SPROCKET
    AREA                    7
    REGION                  2568
    SITE                    74

    FOLLOWING DATA CAN BE AMENDED
    UNIT PRICE             ON ORDER             CURRENT STOCK
         2.45                   786                  178

     *** ENTER DATA FOR UPDATE ***
```

Figure 6-2. Tailored Data Display Screen

To define the screen image layout for the Data Display screen, first write the transaction GENERATE statement using the SPOS=SIMAGE operand. Following this, write the screen image definition: that is, write each line that will appear on the screen exactly the way you want it to appear. Each 80-byte line maps into one line on the screen. Don't use sequence numbers on your screen image lines.

Write out literals exactly as they are to appear. If you want a literal line to be highlighted on the screen, precede the line with a single quotation mark (e.g., 'ENTER NEW PART NO. HERE).

Where data fields are to appear, indicate them using the form:

&nFFFF.XX

where:

& is a delimiter

n is the field mode. The field mode can be:

4 - modifiable in transaction modes 1-6
5 - modifiable in transaction modes 1-5
6 - non-modifiable
7 - modifiable, but non-displayable

FFFF is the field ID

XX is the segment ID qualifier, which can be omitted only if the field ID is unique in this Rules Generator run.

DISPLAY and MODE operands on FIELD or SEGMENT statements are ignored when screen image is used.

Data fields used in a screen image must be in data base or pseudo segments included in the transaction via the DBPATH or TSEGS operands of the GENERATE statement.

You must include the following system fields in your screen image:

&OPTION The OPTION field. Abbreviation &O.

&SYSMSG The 70-character system message field (IMSADF II messages also appear here).

You should include the following system fields unless you included DTRAN=NO or DKEY=NO on the preceding GENERATE statement:

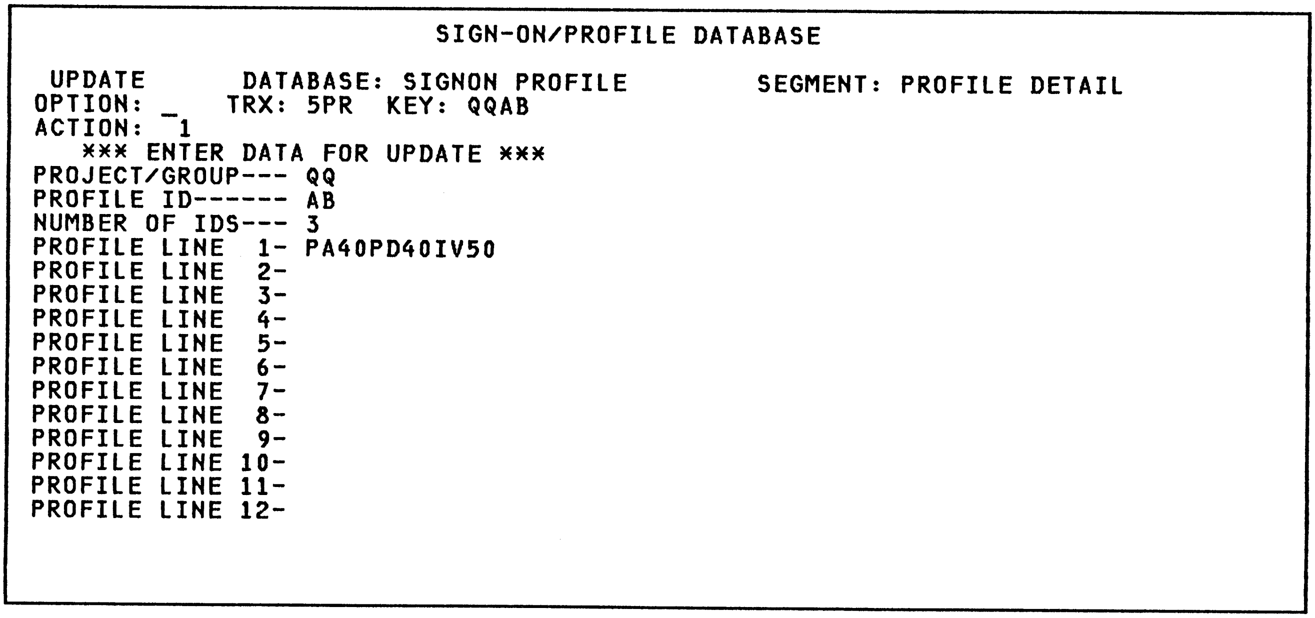
&TRAN The three-character transaction mode and ID. Abbreviation &T. Required unless DTRAN=NO is coded on the GENERATE statement.

&KEY The fully concatenated key field. The default length is 50, but it can be altered (from 1 to 100) by the MAXKEY operand of the GENERATE or SYSTEM statement. Required unless DKEY=NO is coded on the GENERATE statement.

The Rules Generator will place the system fields mentioned above on the last two lines of the screen if you do not include them elsewhere. Hence, if you code other fields on the last two lines and omit these required fields, you will receive error messages that indicate overlapping fields.

PHYSICAL PAGING

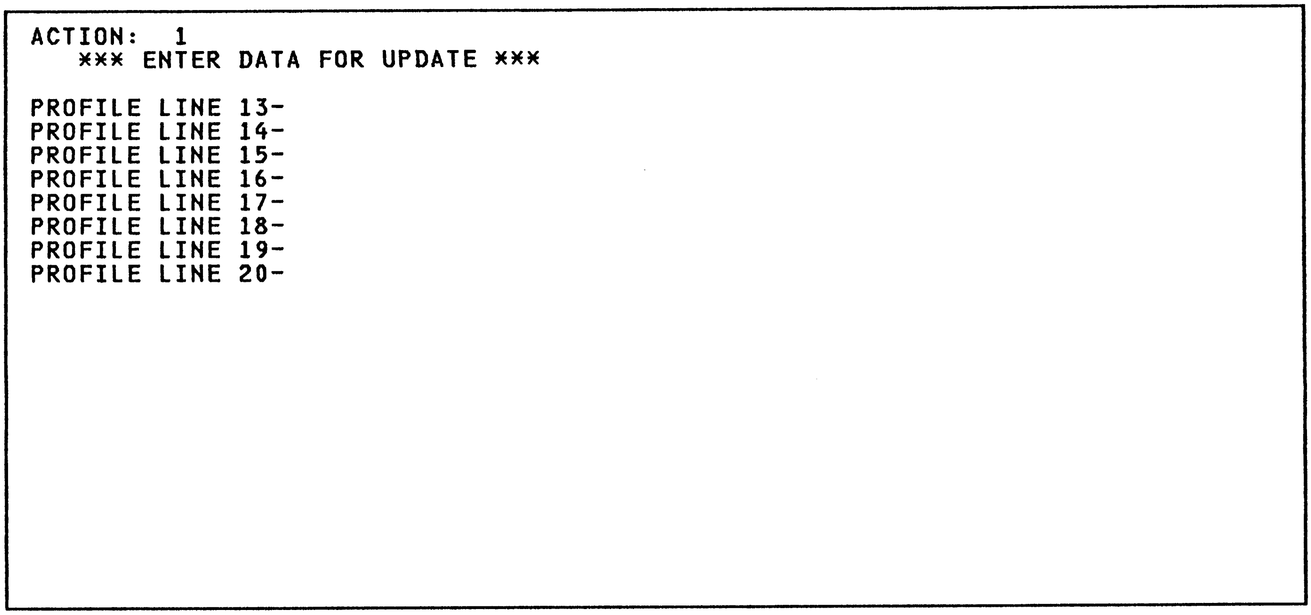
You can request physical paging of a Data Display screen if you use screen image. This is useful when you want to implement a transaction that needs more than one screen of data. Transaction PR (which maintains the security profiles), in application system MFC1, contains an example of physical paging. Figure 6-3 and Figure 6-4 show the two pages. The user receives the first page after key selection, performs amendments, and presses ENTER to receive the second page.



```
                          SIGN-ON/PROFILE DATABASE

 UPDATE      DATABASE: SIGNON PROFILE        SEGMENT: PROFILE DETAIL
OPTION: _   TRX: 5PR  KEY: QQAB
ACTION: 1
   *** ENTER DATA FOR UPDATE ***
PROJECT/GROUP--- QQ
PROFILE ID------ AB
NUMBER OF IDS--- 3
PROFILE LINE  1- PA40PD40IV50
PROFILE LINE  2-
PROFILE LINE  3-
PROFILE LINE  4-
PROFILE LINE  5-
PROFILE LINE  6-
PROFILE LINE  7-
PROFILE LINE  8-
PROFILE LINE  9-
PROFILE LINE 10-
PROFILE LINE 11-
PROFILE LINE 12-
```

Figure 6-3. First Page of PR Transaction Display

On page 2 the user may write further data and then press ENTER again. If there is another page, it will be displayed. The transaction is scheduled to accept the data when the user enters the last page, when he places the value E1 into the ACTION field and presses ENTER on any page, or when he presses PF key 4 on any page.



```
ACTION:  1
   *** ENTER DATA FOR UPDATE ***

PROFILE LINE 13-
PROFILE LINE 14-
PROFILE LINE 15-
PROFILE LINE 16-
PROFILE LINE 17-
PROFILE LINE 18-
PROFILE LINE 19-
PROFILE LINE 20-
```

Figure 6-4. Second Page of PR Transaction Display

The user can return to the first page by typing R1 in the ACTION field and pressing ENTER. However, if this is done, any data that has been entered is lost. The first page is redisplayed without any amendments that may have been made to it.

To request physical paging, code a screen image definition using the control symbol **&=P** starting in column 1 to mark the start of the next physical page. Each page must then include the system ACTION field, defined as &ACTION (the three-character ACTION field).

Each page can include a &SYSMSG field. However, if this is done, the same data field cannot appear on multiple pages (unless you move data into pseudo segments to get around this) and the &OPTION, &TRAN, and &KEY fields must be on the first page.

OTHER CONTROL SYMBOLS

&* in column 1 denotes a comment line
&=nn marks nn blank lines
&: begins the definition of a short field

Sometimes the definition of a field takes up more space than the field itself and so the screen is artificially restricted because there is not enough room on a line to accommodate the field definition. In such cases, define the fields in a fixed column, or tabular, format at the end of the screen or physical page image in which they are to appear (before &=P or &ENDS).

Restriction: Don't mix ordinary screen image definitions and tabular definitions on the same line. Define each line entirely in screen image or entirely in tabular format.

The tabular format of small field definitions can take time to set up because you must make sure that everything is in the correct columns. Therefore, if you are using a time sharing system, you should prepare a small file with commented headings and delimiters marked out. This will make it easier to find the correct columns. See Figure 6-5 for an example. Once your file is prepared, you can merge it into your screen image source file using an editor.

```
&*         1     1     2      3      3       4    5   5
&*3--------2-----8-----4------1------8-------6----1---5
&*    ID  *SLEN *VROW *VCOL  *VMODE *ASTATUS*KSEL*CLR*HLT
&::ffff    :6    :3    :25    :5     :PA     :R   :   :
```

Figure 6-5. Defining Fields Using a Tabular Format

Note: An operand value may be started in any column within the range specified as long as it is completed within the same range.

Columns **Operand** **Description**

3-10 ID=ffff
ffff.xx
name
Identifies the field either by field ID and segment ID or by NAME operand value on the FIELD statement.

12-16 SLEN=xxxx
Display length of field.

18-22 VROW=xx
Row on which field is to be displayed.

24-28 VCOL=xx
Start column of the displayed field.

VCOL=SYMBOLIC NAME
A symbolic reference name may be used in place of row and column entries. The reference name (1-6 characters with first character alphabetic) is used in the screen image to specify that application field location and then referenced by the entry in the column value positions. The row entry is left blank. For example:

```
*
SCREEN IMAGE HEADING
SCREEN LITERAL FOR FIELD UNIT.  XX-----&A
SCREEN LITERAL FOR FIELD DATE.  XX-----&B
*
&*  ID       *SLEN*VROW*VCOL *VMODE*ASTATUS*KSEL
&: UNIT                   A     5      A
&: DATE.YY                B     6
*
*ENDS
```

This technique allows consecutive short fields to be defined without specifying row and column (e.g., &S&G&AA&BB). Reference field usage takes precedence over ADFNAME usage. If you use &A for a data field, then you cannot use &A for the ACTION field. Instead, you could use &AC or &ACT for the ACTION field.

31-36 VMODE=n
Display mode; **n** can be:

4 displayed, modifiable in transaction modes 1-6

5 displayed, modifiable in transaction modes 1-5

6 displayed, not modifiable

7 not displayed, but modifiable

Columns	**Operand**	**Description**

38-44	ASTATUS=FRPA	Specifies audit parameters that apply to this field on this transaction. The following parameters may be included in the generated Input Transaction Rule and can be entered in any order.
		F = FAUDIT=Y
		R = REQUIRED=Y
		P = PREAUDIT=Y
		A = AFA=Y
		Refer to the FIELD operands in the IMS Application Development Facility II Version 2 Release 2 Application Development Reference for a description of these parameters.

46-49	KSEL=z	KSEL specifies key selection phase options which override those on the FIELD statement for this field in this particular transaction. Possible values for **z** are: **N** Not a related field for this transaction **K** Key field displayed without auditing **KA** Key field displayed with auditing **KN** Key field audited but not displayed **KP** Key field displayed with primary auditing only **KS** Key field displayed with secondary auditing only **R** Related field without auditing **RS** Related field with secondary auditing
51-53	CLR=x	Specifies color. CLR is valid only if DEVTYPE=6 or 7 is coded on the GENERATE statement. Possible values for **x** are: **B** Blue **R** Red **P** Pink **G** Green **T** Turquoise **Y** Yellow **W** White
55	HLT=y	Requests extended highlighting. Possible values for **y** are: **D** Default (no extended highlighting) **B** Blink **R** Reverse **U** Underscore

STORING SCREEN IMAGE DEFINITIONS

Instead of defining screen images directly in line with the rules, you can store them in a library and call them in using the IMAGE operand of the transaction GENERATE statement. The Rules Generator JCL must include a DD card with ddname IMAGELIB pointing to the screen image library. The advantages of doing this are:

- It allows sequence numbering of other Rules Generator input
- It makes it easy to rerun Rules Generator statements with * signs in column 1 preceding GENERATE statements (thus rendering them as

comments), a common practice when altering a few details for an application to avoid regenerating unnecessarily large numbers of rules.

The IMAGE operand names the library member that contains the screen image for the transaction.

For example:

```
  GENERATE  TRXID=ST,TRXNAME='STOCK MAINT',DBPATH=IV,
            OPT=CVALL,SPOS=SIMAGE,IMAGE=SISAMPST
```

PROGRAM FUNCTION KEYS

By default in IMSADF II, program function (PF) keys 1, 2, and 3 are used for logical paging (see Chapter 11, "Nonconversational Processing"), and PF key 4 is used for physical paging.

For IMS/VS MFS, you can define your own PF key usage on the Data Display screen by coding PFKDATA=YES on the Rules Generator transaction GENERATE statement. You also must provide a TSEGS operand to define a pseudo segment. When the user presses a PF key (1-11), the number (01-11) will be placed into the first field in the first pseudo segment named in the TSEGS operand.

If you wish to allow more or fewer than 11 PF keys, code PFKNUMB=n on the GENERATE statement (where **n** is between 1 and 36).

If you require values other than 01 to 36 to be mapped into the pseudo segment field, omit the PFKNUMB operand and instead code PFKLIT=(kk), where **kk** is the PF key literal for one key as described in the _IMS/VS Message Format Service (MFS) User's Guide_. Code one PFKLIT operand for every PF key number you wish to use. MFS permits two forms:

- PFKLIT=('abc')
- PFKLIT=(1='abc')

The advantage of the second form is that PF key numbers do not have to be allotted sequentially. The Rules Generator does not check the validity of the PFKLIT operand; that is done by MFS. The maximum length allowed between the parentheses is 22 characters.

Example

Significant Rules Generator statements:

```
  SEGMENT    ID=PF,TYPE=PS
  FIELD      ID=PFKV,L=2,DISP=NO,AUDIT=YES     (Receives PF key values)
  GENERATE   TRXID=PD,DBPATH=PD,OPTIONS=CVALL,
             TRXNAME='STANDARD INFORMATION',
             TSEGS=PF,PFKDATA=YES,PFKNUMB=24,
             DEVNAME=(A3),DEVTYPE=(3)
```

The number (01 to 24) is placed in the PFKV field when the user presses a PF key after viewing a Data Display screen. Audit rules or a special processing routine can then act on the value, perhaps by initiating a transaction switch.

SIGN-ON SCREEN

You can tailor the layout of the Sign-On screen for an application system. You may also create a single Sign-On screen that applies to all IMSADF II application systems. Do this by including the SYSID system field on the screen image. The user can then enter it at the same time as he enters sign-on information. If SYSID is omitted from the image, the default is the first four characters of the MOD name used with the IMS/VS /FORMAT command.

To produce a tailored Sign-On screen using a screen image definition, code the operand SOIMAGE=YES on the Rules Generator GENERATE statement (with OPT=CVSYS); then key in the screen image definition. Only the following system fields may be defined for a Sign-On screen image.

- These are exclusive to the Sign-On screen:

USERID	six-character user ID
PROJECT	one-character project code
GROUP	one-character group code
LOCKWORD	eight-character lockword (password)
SYSID	four-character application system ID

- These are the same as on other screen images:

```
    OPTION     KEY        LTNAME     DATE1     DATE3
    TRAN       SYSMSG     TIME       DATE2     DATE4
```

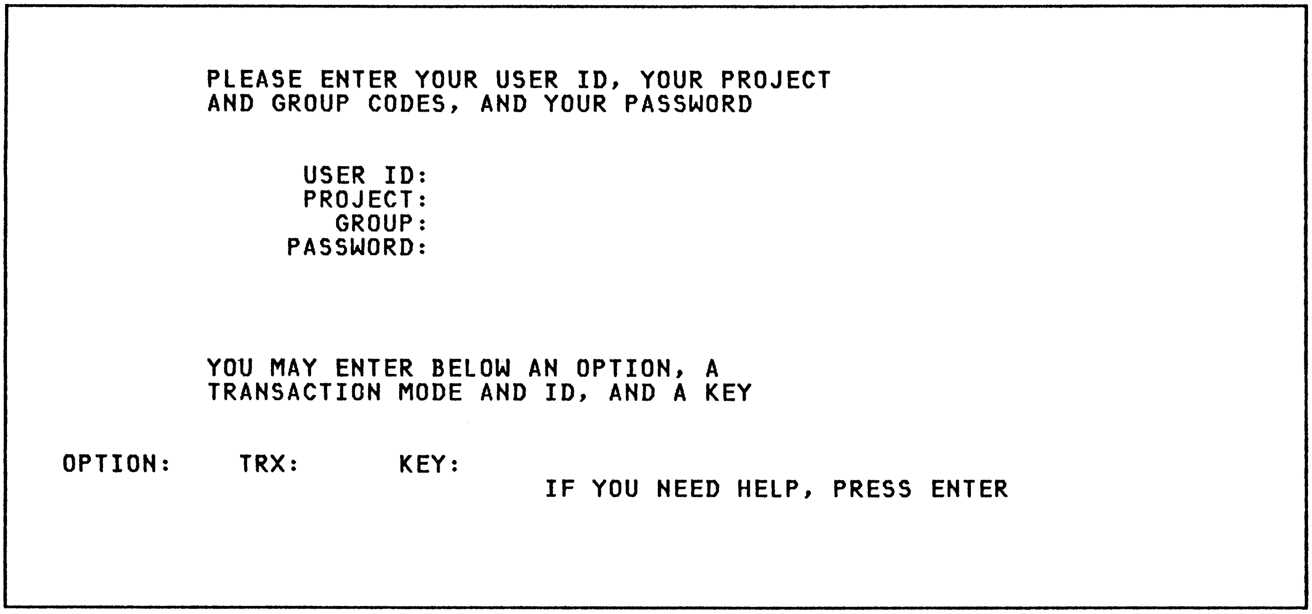
Refer to the SOIMAGE parameter in the IMS Application Development Facility II Version 2 Release 2 Application Development Reference for discussions on field modes.

Example

The GENERATE statement in Figure 6-6 will produce the screen shown in Figure 6-7.

```
         GENERATE OPT=CVSYS,SOIMAGE=YES
&=2
            PLEASE ENTER YOUR USER ID, YOUR PROJECT
            AND GROUP CODES, AND YOUR PASSWORD
&=2
                  USER ID:  &USERID
                  PROJECT:  &PROJECT
                    GROUP:  &GROUP
                 PASSWORD:  &LOCKWORD
&=4
            YOU MAY ENTER BELOW AN OPTION, A
            TRANSACTION MODE AND ID, AND A KEY
&=2
   OPTION: &O TRX: &T   KEY: &KEY
                                 IF YOU NEED HELP, PRESS ENTER
&ENDS
```

Figure 6-6. GENERATE Statement for Tailored Sign-On Screen



```
          PLEASE ENTER YOUR USER ID, YOUR PROJECT
          AND GROUP CODES, AND YOUR PASSWORD


                USER ID:
                PROJECT:
                  GROUP:
               PASSWORD:




          YOU MAY ENTER BELOW AN OPTION, A
          TRANSACTION MODE AND ID, AND A KEY

 OPTION:     TRX:       KEY:
                                 IF YOU NEED HELP, PRESS ENTER
```

Figure 6-7. Tailored Sign-On Screen

TRANSACTION SWITCHING

The terminal user can amend the TRX area on the Data Display screen in order to start another transaction. He can also alter the transaction code (1-6) and the concatenated key. Any of these actions, alone or in combination, will cause a fresh start of the selected transaction. The user is next presented with a key selection or Data Display screen, depending on whether or not the complete concatenated key has been supplied.

You may also write code to request a transaction switch, using the following assignment statements to determine the switching:

```
  TRXID =
  MODE =
  SPAKEYID =
```

There are certain restrictions:

- Only a literal number (1 to 6) can appear to the right of the mode assignment.
- Only a two-character literal enclosed in single quotation marks can appear to the right of the TRXID assignment. If the literal must be assigned from a field (e.g., entered from a screen or derived from a table), an exit routine is needed.
- Only a field name can appear to the right of the SPAKEYID assignment. Logic must be written to set up the concatenated key in a pseudo segment.
- An exit routine is needed to pass data (other than the concatenated key) (see "Use of Exits" on page 6-12).
- Setting SPAKEYID alone will have no effect. MODE and TRXID must also be set to cause a switch, even if their values are not changed.

The TRXID, MODE, and SPAKEYID fields can also be the source of assignment operations.

Appendix E, "Switching Between COBOL and IMSADF II Transactions" shows techniques for switching between IMSADF II transactions and non-IMSADF II COBOL transactions.

Example

To switch from the PD to the IV transaction, the following statements in the high level audit language would be suitable:

```
  TRXID = 'IV'
  SAPSCKEY = SPAKEYID
  SAPSIKEY = '0022A58722'
  SPAKEYID = SAPSCKEY
```

where IKEY and CKEY are in the following pseudo segment:

```
  SEGMENT ID=PS,TYPE=PS
  FIELD   ID=CKEY,L=33
  FIELD   ID=PKEY,L=17,POS=1
  FIELD   ID=IKEY,L=16,POS=18
```

SEQUENCE OF OPERATIONS

When transaction switching is requested through the Auditor, the switch does not take place instantly, but at a predetermined time. Figure 6-8 illustrates the sequence.

Switching can be requested from any of three audit phases (KEY, PRELIM, and PROCESS) and from any of the three legs (P0, P1, P2). The Auditor completes current processing before switching. In particular, this means:

- During primary key audit, the logic specified for the current key will be completed.

- During secondary key audit, the logic specified for the current key will be completed.

- During the PRELIM phase (preaudit), all logic for all fields will be completed (including the message sending), but the data will not be displayed unless an error occurs.

- During the PROCESS phase, all logic for all fields will be completed. If no error occurs, the transaction driver will update the data bases and send any automatic messages and secondary transactions before switching. The confirmation display (SEGMENT MODIFIED SUCCESSFULLY) is bypassed.

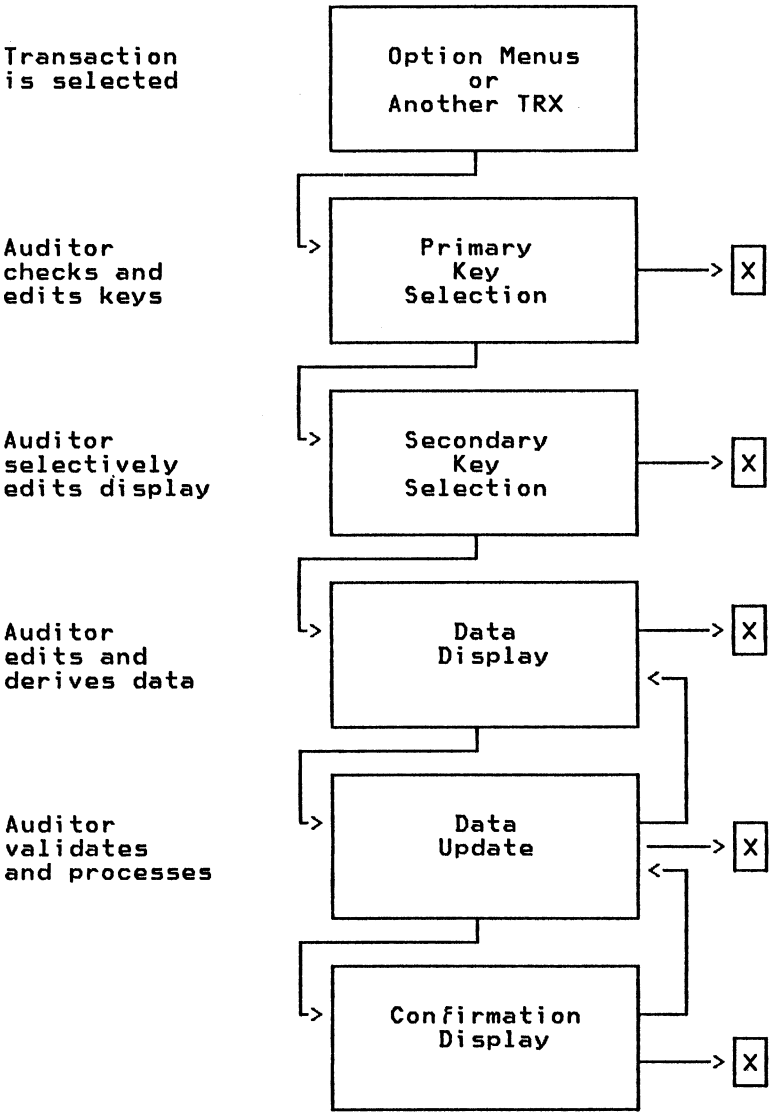

Transaction is selected
Option Menus or Another TRX
Auditor checks and edits keys
Primary Key Selection
X
Auditor selectively edits display
Secondary Key Selection
X
Auditor edits and derives data
Data Display
X
Auditor validates and processes
Data Update
X
Confirmation Display
X

Figure 6-8. Where the Auditor is Invoked

USE OF EXITS

You can enhance the capabilities of the Auditor by writing additional functions in COBOL, PL/I or Assembler. These functions are invoked by the AEXIT statement in the high level audit language.

One use for exits is to pass information, other than the concatenated key, when switching transactions. The program must move the data into a field in the SPA named SPAFLDSG. An example of such a program appears in Chapter 9, "Exits." A similar program is needed in the receiving transaction.

The length available in SPAFLDSG is set by means of the COMMLEN operand on the Rules Generator GENERATE statement (with OPT=CVALL) that defines the transaction. The maximum is constrained by the size of the SPA. Additionally, an installation option can set the communication area at sign-on time. If your installation uses this option, the value you give to the Rules Generator must be as large as you need plus whatever is used at sign-on. (The IMS Application Development Facility II Version 2 Release 2 Installation Guide contains an appendix that shows how to calculate the SPA size.) The program can find out the size of the COMMLEN setting by examining the SPACOMLN field in the SPA.

If you want to switch to a transaction named in a field rather than a literal, you need an exit to move the field value into SPACGTRX in the SPA. A dummy value must also be assigned to the TRXID system field to cause IMSADF II to take notice of SPACGTRX.

For example, you cannot write:

```
  TRXID = SAPSNEWT
```

You must write:

```
  TRXID = 'XX'
  IF AEXIT 71 SAPSNEWT RETURN TRUE
  ENDIF
```

and write exit routine number 71 in COBOL, PL/I or Assembler to move the value of SAPSNEWT, the related field, into SPACGTRX.

MULTIPLE-PATH TRANSACTIONS

As you know, standard processing transactions are defined via the GENERATE statement of the Rules Generator.

For example:

```
  GENERATE  TRXID=PI,TRXNAME='PARTS INFORMATION',
            OPT=CVALL,DBPATH=(CY,OR),TSEGS=(WO,PD)
```

The segments named in the DBPATH operand are defined in preceding SEGMENT statements and are the target segments of the transaction. The data base layouts assumed are shown in Figure 6-9. WO is the ID of a pseudo segment.

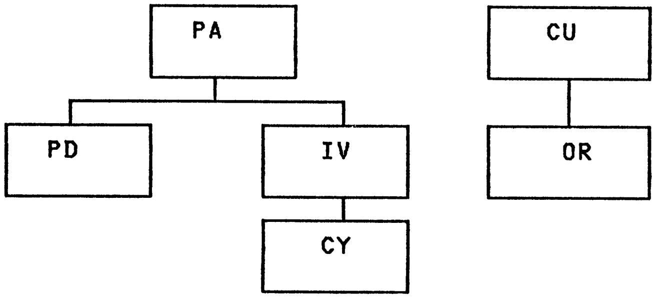


Figure 6-9. Data Bases Used in Examples

If any field from the CU segment is displayed (via the DISP=YES operand on a FIELD or SEGMENT statement or by inclusion in the screen image), the CU segment will be included in the transaction and will be updated if data is changed by the online user or by audit rules. The same applies to the PA and IV segments. If no field from a segment is displayed but audit logic is required to access or update it, include the segment in the DBPATH thus:

```
  DBPATH=(CY,IV,OR)
```

The target segments are still CY and OR because they are the lowest in each hierarchical path. Collectively, the segments named or implied by the DBPATH operand are called DBPATH segments. The user is prompted for their keys by key selection and together their keys constitute the concatenated key of the transaction.

Segments named in TSEGS are either pseudo segments or data base segments to be retrieved by the Auditor under control of audit rules or by a special processing program.

Updating of DBPATH segments is controlled by the transaction mode selected by the user. (Modes 1 and 2 are interchangeable with 3 and 4, respectively.)

Mode 5: The user is prompted by key selection to enter keys of existing DBPATH segments and the segments are displayed. If he changes data, the changed segments will be updated on the data base. If auditing changes data, those segments will also be updated. The Auditor is invoked only if the user changes some data on the screen.

Mode 4: The user is prompted by key selection to enter keys of existing nontarget DBPATH segments but is required to enter the key of at least one target segment that does not exist on the data base so that it can be inserted. For the other target segments he can enter an existing key or one that does not exist. If the user changes data, changed segments are replaced and new segments are inserted. Again, auditing changes also lead to segments being updated (replaced or inserted). The Auditor is invoked whether the user changes some data on the screen or not.

Mode 3: The user is prompted by key selection to enter keys of existing DBPATH segments. For transactions with a single target segment, the Auditor will be called and the segment will be deleted regardless of whether the user changes data on the screen. If the user or the Auditor changes data in other segments, they will be replaced.

For transactions with multiple target segments, mode 3 is just like mode 5, except that in mode 3 the Auditor will be called (and can therefore cause updates) whether the user changes data or not.

DELETE ELIGIBILITY

To define a transaction that deletes segments other than the target segment in a single path transaction, you must:

- Define delete eligibility against those segments

- Code DL/I calls through the audit operation to delete the segments

Use the DLET operand on the transaction GENERATE statement.

In the following example the audit rule checks for a transaction mode of 3 before deleting. The audited field is a nondisplayed dummy field in the pseudo segment WO. The user receives the display with the message **PRESS ENTER TO DELETE DATA**. When he does so, the CY and OR segments are deleted.

Significant Rules Generator statements:

```
SEGMENT   ID=WO,TYPE=PS,LENGTH=1,DISP=NO
FIELD     ID=DUMY,LENGTH=1,AFA=YES
GENERATE  TRXID=OM,TRXNAME='ORDER MAINT',
          OPT=CVALL,DBPATH=(PD,CY,OR),
          TSEGS=WO,DLET=(CY,OR)
```

High level audit language:

```
SYSID = SAMP
SEGID = WO
FIELD = DUMY
IF MODE = 3
  IF DLET KEYFIELD CY NOT OK
    ERRORMSG = 1025
    ENDIF
  IF DLET KEYFIELD OR NOT OK
    ERRORMSG = 1025
    ENDIF
  ENDIF
```

Note: The related field KEYFIELD is a special value recognized by the DL/I call audit operation as meaning the key of the segment already retrieved.

If an attempt is made to delete a segment using the audit operation and the segment is not eligible for deletion, the deletion is not done, the audit operation returns false, and a DL/I status code of AM is set.

The deletion is not performed immediately; the operation merely sets a flag. The transaction driver performs the deletion later when it performs any other data base updates.

Segments named in the DLET operand must be DBPATH or TSEGS segments.

To complete the example, Figure 6-10 shows some fields.

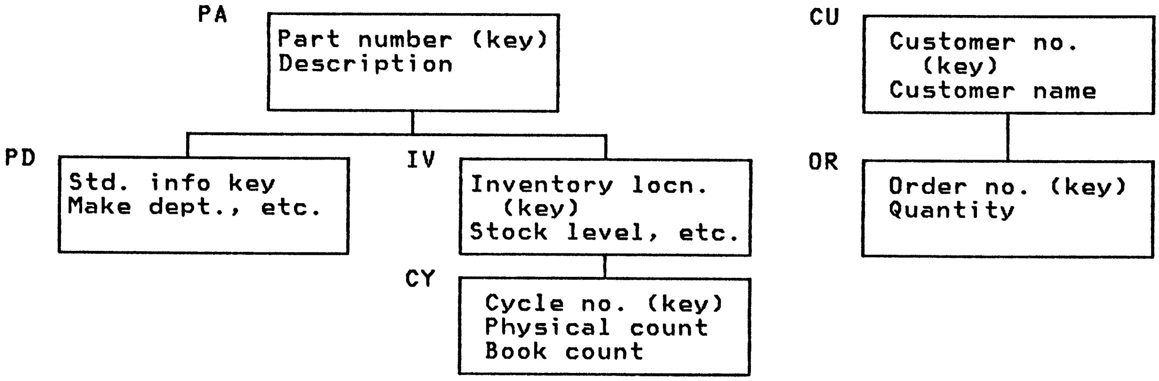
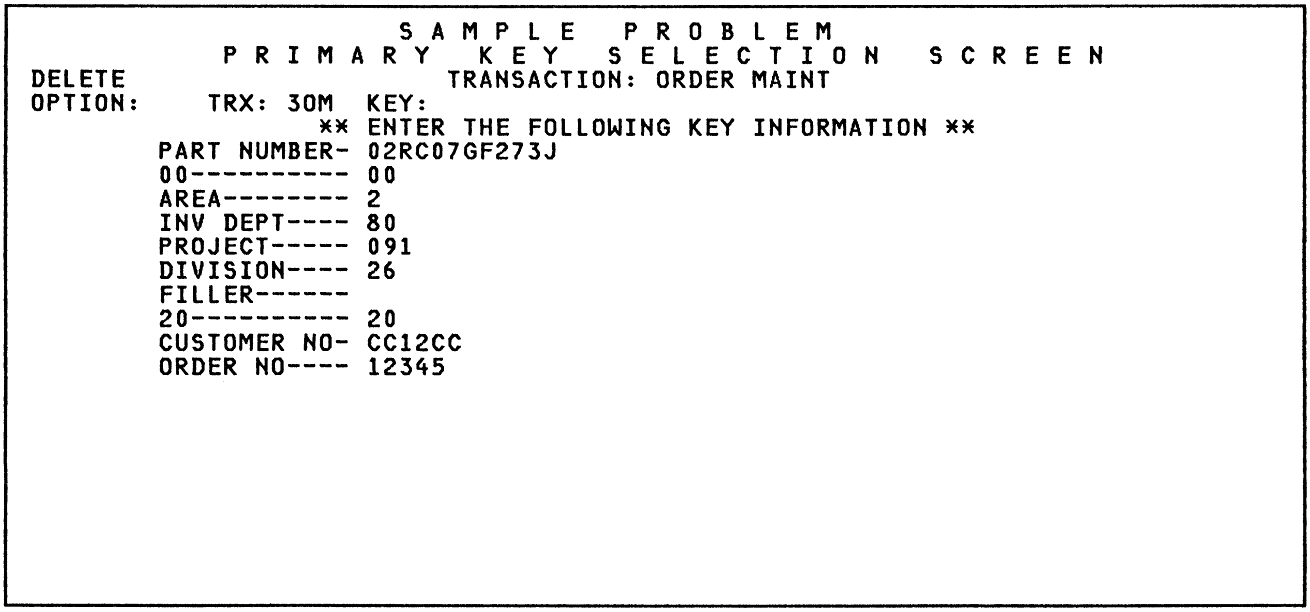


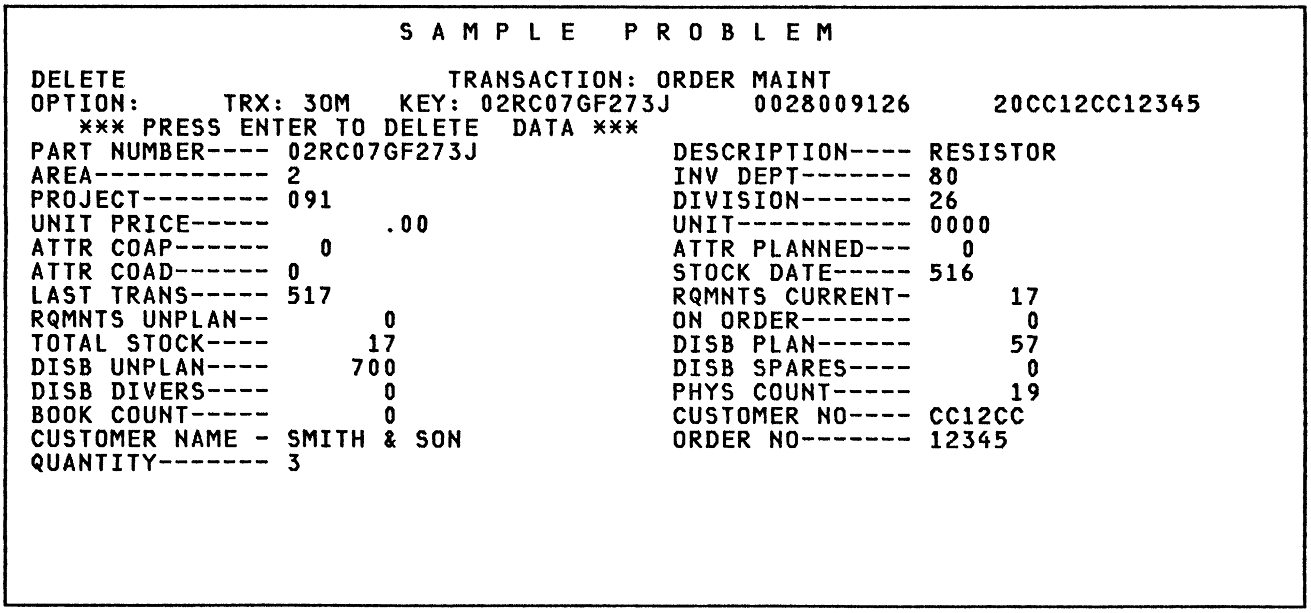
Figure 6-10. Data Base, Showing Some Fields Used on Screens

Figure 6-11 and Figure 6-12 show how the Key Selection and Data Display screens for the order maintenance transaction appear in delete mode.



```
                       S A M P L E   P R O B L E M
            P R I M A R Y   K E Y   S E L E C T I O N   S C R E E N
DELETE                    TRANSACTION: ORDER MAINT
OPTION:    TRX: 30M  KEY:
                  ** ENTER THE FOLLOWING KEY INFORMATION **
        PART NUMBER- 02RC07GF273J
        00---------- 00
        AREA-------- 2
        INV DEPT---- 80
        PROJECT----- 091
        DIVISION---- 26
        FILLER------
        20---------- 20
        CUSTOMER NO- CC12CC
        ORDER NO---- 12345
```

Figure 6-11. Key Selection Screen in Delete Mode



```
                       S A M P L E   P R O B L E M

DELETE                    TRANSACTION: ORDER MAINT
OPTION:     TRX: 30M   KEY: 02RC07GF273J     0028009126     20CC12CC12345
   *** PRESS ENTER TO DELETE  DATA ***
PART NUMBER---- 02RC07GF273J            DESCRIPTION---- RESISTOR
AREA----------- 2                       INV DEPT------- 80
PROJECT-------- 091                     DIVISION------- 26
UNIT PRICE-----       .00               UNIT----------- 0000
ATTR COAP------   0                     ATTR PLANNED---   0
ATTR COAD------ 0                       STOCK DATE----- 516
LAST TRANS----- 517                     RQMNTS CURRENT-      17
RQMNTS UNPLAN--       0                 ON ORDER-------       0
TOTAL STOCK----      17                 DISB PLAN------      57
DISB UNPLAN----     700                 DISB SPARES----       0
DISB DIVERS----       0                 PHYS COUNT-----      19
BOOK COUNT-----       0                 CUSTOMER NO---- CC12CC
CUSTOMER NAME - SMITH & SON             ORDER NO------- 12345
QUANTITY------- 3
```

Figure 6-12. Data Display Screen in Delete Mode

INSERT ELIGIBILITY

Insert eligibility has nothing to do with DL/I insert calls from audit operations which can be coded regardless of insert eligibility. Insert eligibility alters key selection and data base updating to allow insertions of DBPATH segments (target or not) in any transaction mode except 6. Segments are made eligible through the ISRT operand of the transaction GENERATE statement.

Mode 6: No updates are performed in this mode. However, if a key is not entered for a DBPATH segment that has insert eligibility, and if no occurrences exist for that segment, key selection will proceed to the Data Display screen.

Mode 5: During key selection the user is prompted to enter keys of DBPATH segments. If a segment is eligible for insertion, he is free to enter a key that does not exist on the data base. If he does and proceeds to enter data into it, the segment will be inserted. For an existing key, the segment will be replaced.

Mode 4: The restriction on inserting several segments in a path does not apply. If an eligible segment is to be inserted, all its dependents can be inserted at the same time, whether marked for eligibility or not.

Mode 3: As in mode 5, the eligible segments can be inserted or replaced depending on the keys entered by the user.

In the order maintenance example (Figure 6-10, Figure 6-11, and Figure 6-12), the transaction generated (without the ISRT operand) will behave as follows:

Mode 5: Any of the six segments can be replaced by amendments from the terminal.

Mode 4: The PA, IV, and CU segments can be replaced while the target segments PD, CY, and OR can be inserted or replaced, depending on the keys entered by the user, but provided at least one is inserted.

Mode 3: Audit operations will delete CY and OR. Other segments will be replaced if changed by the user.

If you code ISRT=PD on the GENERATE statement, it will behave as follows:

Mode 5: Any segment can be replaced. The PD segment can be inserted or replaced.

Mode 4: The ISRT operand makes no difference.

Mode 3: CY and OR will be deleted. PD may be inserted or replaced. PA, IV, and CU may be replaced.

If you code ISRT=(PA,PD) on the GENERATE statement, it will behave differently. Such a transaction is unlikely to be used in any mode except 4:

Mode 4: The PA, PD, IV, CY, and OR segments can all be inserted at once while the CU segment can also be replaced.

For a transaction like this (insert eligibility on a high level segment), you should impose a security level of 4 in the Sign-On Profile data base and educate the user to use mode 4 only. View the transaction as 4OM, rather than as OM in mode 4.

DL/I CALLS FROM THE AUDITOR

This section explains how to perform DL/I calls under control of audit rules and when to use this function. The main uses are:

- Processing twins (multiple segment occurrences)
- Validating data received from the screen against other data bases (e.g., valid customer number entered)
- Controlling segment deletions (this is the only way to delete segments other than the target segment of a single path transaction)
- Controlling segment insertions (used as a supplement or an alternative to using insert eligibility)
- Retrieving or updating segments without key or search fields that identify them uniquely

HOW THE DL/I CALL OPERATION WORKS

Segments to be retrieved by a DL/I Auditor call must have space reserved for them in the segment area (within the SPA). This is done by means of the TSEGS operand of the GENERATE statement. The layout shown in Figure 6-13 is the result of the following transaction definition based on the sample data base (Figure 6-9 on page 6-12):

```
  GENERATE  TRXID=IM,TRXNAME='INVENTORY',
            OPT=CVALL,DBPATH=PD,
            TSEGS=IV
```

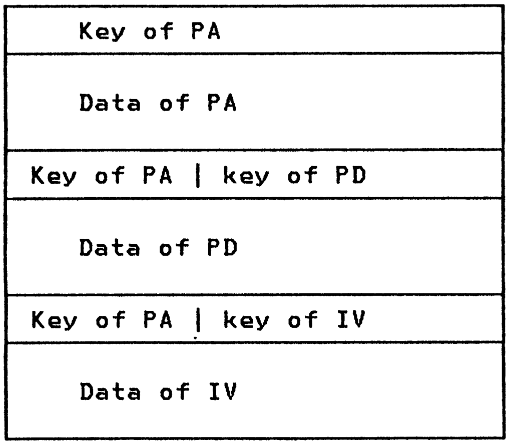


Figure 6-13. Layout of the Segment Area (in the SPA)

As shown, one place is reserved for each segment type. (Twin processing is explained in "Multiple Segment Occurrences (Twins)" on page 6-25.) IMSADF II keeps the concatenated key of each segment separately. The user is prompted for the keys of the DBPATH segments through key selection. At preaudit time, therefore, the concatenated keys of PA and PD are already set up and the segments are loaded. The concatenated key of IV is not defined - not even the root key portion - and must be supplied by audit rule processing before the segment can be retrieved. As shown in earlier examples, the related field coded in the DL/I call statement contains the concatenated key.

Whenever a segment is successfully retrieved, whether through key selection, by a DL/I Auditor call, or by means of special processing, the concatenated key is set up in the appropriate area as illustrated in Figure 6-13.

The concatenated key of a segment consists of the concatenated key of its parent, followed by the key of the segment itself. After a DL/I call, the concatenated key of the segment is returned by DL/I in an area area known as the key feedback area (in the PCB). From the KEY=YES operands on the Rules Generator FIELD statements IMSADF II determines the length of the parent's concatenated key within the key feedback area and moves it into the IMSADF II concatenated key area for that segment ID. It then moves the key of the segment itself from the actual segment data area.

Segment Flags

In addition to the key and the data, IMSADF II maintains three flags associated with each data base segment. These are the delete, retrieved, and changed flags. They determine the update processing performed by the IMSADF II transaction drivers. They can be set by the indicator setting statements in the high level audit language.

For example:

```
  PD CHGEFLAG = ON
  PD RTRVFLAG = OFF
  CY DLETFLAG = ON
```

In the above example, segment PD has its changed flag set on and its retrieved flag off, while CY has its delete flag set on.

Figure 6-14 summarizes the action of the transaction driver when processing the segment flags and performing data base updates. (See "Multiple-Path Transactions" on page 6-12 for a review of the treatment of the target segment in transactions that have a single target segment.)

Changed Flag	**Retrieved Flag**	**Delete Flag**	**Resultant Action**
OFF	OFF	OFF	No action
OFF	OFF	ON	No action / Error
OFF	ON	OFF	No action
OFF	ON	ON	Segment deleted
ON	OFF	OFF	Segment inserted
ON	OFF	ON	No action / Error
ON	ON	OFF	Segment replaced
ON	ON	ON	Segment deleted

Figure 6-14. Segment Flag Processing

The DL/I update calls issued through the high level audit language without the immediate (IMMED) option merely set these flags; they do not cause the segments to be updated until the transaction driver reaches that point in its processing.

After an update has taken place, the flags are reset to avoid a repetition of the call. Figure 6-15 shows the flag settings after a successful update has been performed.

Action	Resultant Settings		
	Changed Flag	Retrieved Flag	Delete Flag
Segment deleted	OFF	OFF	OFF
Segment inserted	OFF	ON	OFF
Segment replaced	OFF	ON	OFF

Figure 6-15. Flag Settings after a Successful Update

The flags will be reset in this way after any update; that is, whether caused by a DL/I Auditor call with the IMMED option, by a special processing or audit exit routine, or by the transaction driver itself as a result of the flag settings (normally after auditing is complete).

DL/I CALL EXPRESSIONS

The format of a DL/I call in the high level audit language is as follows:

```
  IF function <IMMED> keyfield segid <NOT> OK
```

The possible functions are the DL/I call functions explained in the next section.

The optional keyword IMMED is used in association with update functions (ISRT, REPL, HREP, DLET, HDEL) to indicate that the function is to be performed immediately. Otherwise, the operation is performed later by the transaction driver.

The keyfield is the name of a field containing the key to be used in the operation. If the special name KEYFIELD is used, this means: use the key value already saved by IMSADF II in the concatenated key area.

The segid is the ID of the segment against which the operation is to be performed.

OK or NOT OK is coded to determine the next statement to be performed, depending on the outcome of the DL/I operation. If the operation is is successful (blank status code) and OK is coded, the statement after the IF will be performed.

NOTE: VSAM files (KSDS) in the IMS/VS environment are treated like root only DL/I data bases. In the CICS/OS/VS environment, IMSADF II simulates the DL/I interface using CICS file control commands. Irrespective of the environment, the developer's view for the support of VSAM files is the same as that for a root only DL/I data base. DL/I calls against VSAM files defined either in DBPATH or TSEGS can be coded in HLAL. The format of the DL/I call expression is the same as above for VSAM files except that only the following DL/I calls can be coded for VSAM files:

DLET
GHN
GHNQ
GHU
GHUU
GN
GNQ
GU
GUU
GU1
HDEL
HREP
ISRT
REPL
SGN

For ESDS files in the CICS/OS/VS environment, only the ISRT command can be coded.

THE DL/I CALL FUNCTIONS

The exact use made of the concatenated key in the DL/I call, whether it is in a related field or KEYFIELD, depends on the function being performed. The main functions are listed below.

Function	Meaning	Description
GU	Get Unique	Uses the entire concatenated key to retrieve the segment with key equal to the one specified and under parents with keys equal to those specified. If two segments have the same key value, this function always retrieves the first.
GUU	Get Unique Unqualified	Uses only the parent portion of the concatenated key to retrieve the first segment occurrence under parents with keys equal to those specified.

Function	Meaning	Description
GN	Get Next	Use after a GU or GUU call to retrieve the next occurrence of the same segment type, or the first occurrence of a dependent segment type. The call uses only the parent portion of the concatenated key, which will normally be the same as that used in a previous, successful GU or GUU call. The system will only move forward in the data base to satisfy the call, but will never go beyond the parent having the specified key. The GN function only works within one execution of the transaction. If the audits finish, the screen is displayed, the user enters amendments and more auditing is performed, the data base position should be re-established with a GU call before a GN is issued. IMSADF II retains the concatenated key across such steps in a conversation, but DL/I loses its position in the data base.
GNQ	Get Next Qualified	Uses the entire concatenated key to retrieve a segment with key equal to the one specified under parents with keys equal to those specified. Like GN, it moves forward in the data base and should be preceded by a GU or GUU call. It is a way of retrieving segment occurrences that have the same (non-unique) keys.
ISRT	Insert	Uses only the parent portion of the concatenated key to insert the segment under the parent with key equal to that specified. The segment being inserted contains its own key. If it is a concatenated segment, it will contain the logical parent's key twice and they must be equal (DL/I requirement). **With IMMED option:** If the IMMED (immediate) option is used on the DL/I call statement, the operation is performed at once. **Without IMMED option:** The segment is not inserted immediately. It is flagged and inserted later by the transaction driver. The Auditor sets the segment's concatenated key in the segment area from the value in the key field named in the DL/I call. It turns the segment retrieved flag off and the changed flag on. Hence, it is not possible to insert multiple occurrences of the same IMSADF II segment ID without the IMMED option. Segment aliases must be defined for this purpose. (See "Multiple Segment Occurrences (Twins)" on page 6-25.)

Function	Meaning	Description
DLET	Delete	Uses the entire concatenated key to delete the segment with key and parent keys equal to those specified. The DLET audit operation will return false and will not be performed unless the segment is eligible for deletion (via the DLET operand of the GENERATE statement for the transaction). The Auditor will return a status code of AM when DLET eligibility has not been specified. **With IMMED option:** The operation is performed immediately. It must be preceded by a Get Hold call (GHU, GHUU, GHN, or GHN). This is a DL/I requirement. That call supplies the key information for the deletion. Do not code a key field name other than KEYFIELD with this operation (it will be ignored). The HDEL call (with IMMED option) combines a GHU with a DLET IMMED. **Without IMMED option:** A delete flag is set and the actual operation is performed later. Do not code a key field name other than KEYFIELD with this operation (it will be ignored). The concatenated key in the IMSADF II segment area will always be used. The transaction driver performs a Get Unique call with the Hold option (GHU) immediately prior to the actual DL/I DLET call (this is a DL/I requirement).
HDEL	Delete with DL/I Get Hold	Uses the entire concatenated key to delete the segment with key and parent keys equal to those specified. The DLET audit operation will return false and will not be performed unless the segment is eligible for deletion (via the DLET operand of the GENERATE statement for the transaction). The Auditor will return a status code of AM when DLET eligibility has not been specified. **With IMMED option:** The operation is performed immediately. First, a DL/I GHU (Get Hold Unique) call is issued, using the key supplied in the DL/I call statement. That call supplies the key information for the delete (DL/I DLET) call, which is performed immediately. This call normally is used to delete a segment that has not previously been retrieved. **Note:** If the segment has been previously retrieved KEYFIELD will always be used. Otherwise, the concatenated key passed in the key field will be used. To override the key of an HDEL call to a retrieved segment, turn off the RETRIEVE flag prior to the call. **Without IMMED option:** The HDEL call without the IMMED option is identical to the DLET call.

Function	Meaning	Description
REPL	Replace	There is seldom a need to code this function as the transaction driver will automatically replace segments retrieved by the Auditor and by special processing programs if they are changed. **With IMMED option:** The operation is performed immediately. It must be preceded by a Get Hold call (GHU, GHUU, GHN, or GHN). This is a DL/I requirement. That call supplies the key information for the replace. Do not code a key field name other than KEYFIELD with this operation (it will be ignored). The HREP call (with IMMED option) combines a GHU with a REPL IMMED. **Without IMMED option** If the REPL call is coded, it uses the entire concatenated key, but the segment changed flag is turned on. Do not code a key field name other than KEYFIELD (it will be ignored). The transaction driver performs the GHU and REPL calls to DL/I later.
HREP	Replace with DL/I Get Hold	The transaction driver will automatically replace segments retrieved by the Auditor and by special processing programs if those segments are changed or marked as changed. It may sometimes be useful to do it under control of audit rules. This will be true if further DL/I operations are needed against the same segment ID before the transaction driver can perform the update (i.e., before the Auditor terminates). **With IMMED option:** The operation is performed immediately. First, a DL/I GHU (Get Hold Unique) call is issued, using the key supplied in the DL/I call statement. That call supplies the key information for the replace (DL/I REPL call), which is performed immediately. **Note:** If the segment has been previously retrieved KEYFIELD will always be used. Otherwise, the concatenated key passed in the key field will be used. To override the key of an HREP call to a retrieved segment, turn off the RETRIEVE flag prior to the call. **Without IMMED option:** The HREP call without the IMMED option is identical to the REPL call.

Other DL/I call functions are available. You can use the Get Next within Parent (GNP) function, but its operation is equivalent to GN because of the way IMSADF II uses the parents' keys. (All SSAs above the lowest level are qualified.)

Two others may be of use in certain circumstances:

Function	Meaning	Description
GU1	Get Unique First	Does not use the concatenated key. (Code KEYFIELD as the related field.) Retrieves the first occurrence of the segment in the data base.
SGN	Sequential Get Next	Does not use the concatenated key. (Code KEYFIELD as the related field.) Use after GU1, GU or GUU. Retrieves the next occurrence of the segment in the data base, crossing hierarchical boundaries as necessary.

DL/I STATUS CODES

All the retrieval operations return true if the segment is retrieved and false otherwise. Similarly, all the update operations (REPL, HREP, DLET, HDEL, ISRT) with the IMMED option return true if the operation succeeds (non-blank status code from DL/I) and false otherwise. Delete calls without the IMMED option return true if the segment is eligible for deletion and false otherwise. Insert and replace calls without the IMMED option always return true. Invalid DL/I status codes during data base updates are the responsibility of the transaction driver unless the IMMED option is used, in which case the programmer must handle them, usually by issuing the ROLL call.

After unsuccessful Get calls, it will be necessary to check DL/I status codes to determine if the failure was due to a normal condition (such as "segment not found") or an abnormal condition (resulting from an error in a rule or an IMS/VS DBD or PSB).

The reserved name STATCODE in the high level audit language can be used to test a DL/I status code.

For example:

```
  IF STATCODE = 'GE'
```

A list of status codes can be quoted, separated by commas. (See the high level audit language coding in the example below.)

The most common DL/I status codes are listed below. A full list appears in the IMS/VS Application Programming Reference Manual (SH20-9026).

Code	Cause
GE	Segment not found. Can occur after any Get call.
GB	End of data base encountered and segment not found. Can occur after Get Next calls.
GA	Segment found but hierarchical boundary crossed when using SGN function. DL/I Auditor call operation returns true in this case.
AC	Segment NAME or PARENT operands in Rules Generator statements inconsistent with PCB.
AK	Field name or segment KEYNAME operands in Rules Generator statements inconsistent with FIELD statements in DBD.
AM	Segment sensitivity cr processing options in PCB inconsistent with call. This can be caused by allowing path calls at the PCB level but restricting them on certain segments. AM can also be received if DLET call is issued against a segment that has no DLET eligibility.

The DL/I status code can be included in an error message by using the name VARLIST1 as described under "Error Messages" on page 4-16.

Example

- Read all twin occurrences of a target segment (TS) and accumulate a field (VALU).

- Upon completion, insert a new segment (NS) containing the accumulated total in another data base.

- The key of the new segment is derived from a field in the root segment (FLD1).

Figure 6-16 shows the data base used in this example.

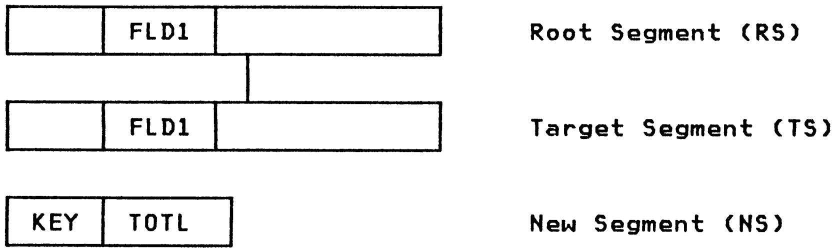


Figure 6-16. Data Base for DL/I Call Example

Significant Rules Generator statements:

```
  SEGMENT   ID=RS,...
  FIELD     ID=FLD1,PAUDIT=YES,...

  GENERATE  TRXID=AC,TRXNAME='ACCUMULATION',
            OPT=CVALL,DBPATH=TS,
            TSEGS=NS
```

High level audit language:

```
      SYSID = SSSS
      SEGID = RS
      FIELD = FLD1
      * MOVE TARGET SEGMENT FIELD TO TOTAL
      SSNSTOTL = SSTSVALU
      * LOOP ACCUMULATING TOTAL
      GETLOOP:IF GN KEYFIELD TS OK
        SSNSTOTL = SSNSTOTL + SSTSVALU
        GOTO GETLOOP
        ENDIF
      IF STATCODE ¬= 'GE,GB'
        ERRORMSG = 7777
        ENDIF
      SSNSKEY = FLD1
      IF ISRT SSNSKEY NS OK
        NOP
        ENDIF
```

SELECTING THE PCB

By default, a DL/I call against a segment will use the PCB indicated by the value of the PCBNO operand on the Rules Generator SEGMENT statement (or on the SYSTEM statement). If it is necessary to reset the PCB number dynamically, code the statement:

```
    xx   PCBNUM = yyy
```

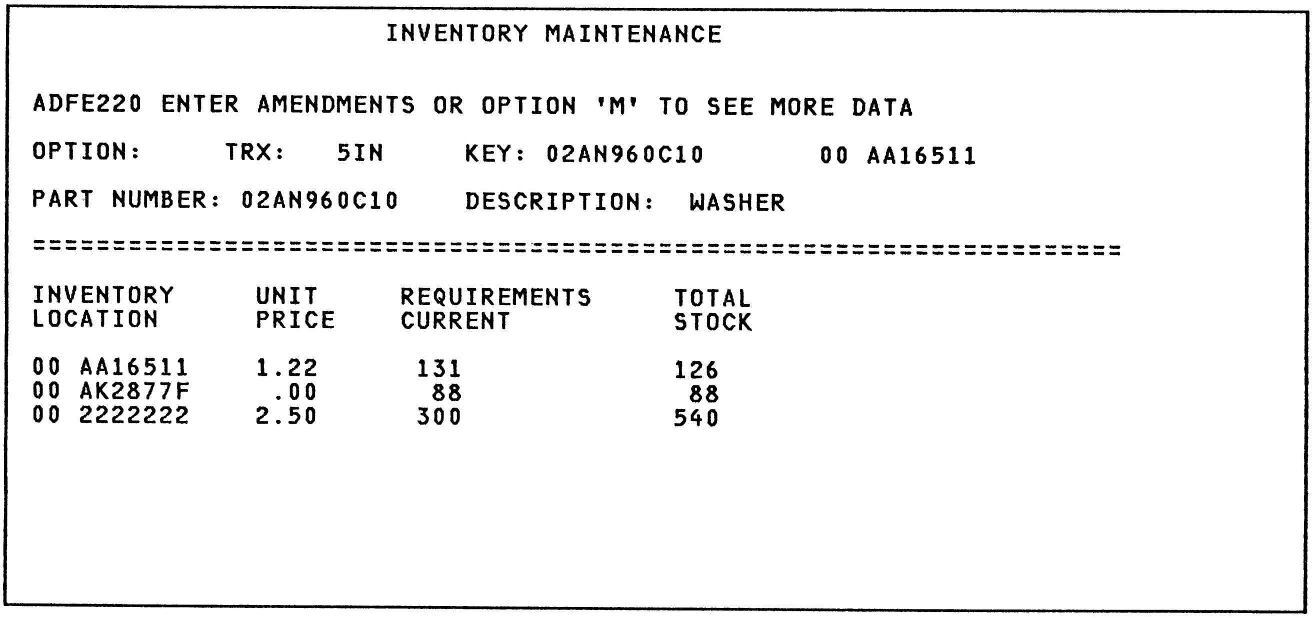
where xx is the segment ID and yyy is the PCB number (maximum 120) relative to the first application data base PCB in the PSB. The number **yyy** can be a literal or a field containing a numerical value.

The new PCB number will be applied not only to the specified segment xx but to every segment in the transaction that uses the same PCB as xx. The change will remain in force until a new IMSADF II transaction starts (i.e., when the user changes OPTION, TRX or a key on the screen or when the Auditor or a special processing program causes a transaction switch) or until another PCBNUM statement resets it.

MULTIPLE SEGMENT OCCURRENCES (TWINS)

Sometimes it is necessary to allow a user to display and update multiple occurrences of a segment on the same screen. The TWINS keyword on the GENERATE OPT=CVALL statement can be used to do this, as the following example shows.

Using the sample data base, a transaction to display and update multiple occurrences of the IV segment will be defined. The Data Display screen is illustrated in Figure 6-17.



```
                        INVENTORY MAINTENANCE

ADFE220 ENTER AMENDMENTS OR OPTION 'M' TO SEE MORE DATA

OPTION:     TRX:   5IN     KEY: 02AN960C10       00 AA16511

PART NUMBER: 02AN960C10    DESCRIPTION:  WASHER

=====================================================================

INVENTORY     UNIT      REQUIREMENTS     TOTAL
LOCATION      PRICE     CURRENT          STOCK

00 AA16511    1.22       131             126
00 AK2877F     .00        88              88
00 2222222    2.50       300             540
```

Figure 6-17. Data Display Screen for Twin Processing Transaction

This screen design allows for three occurrences of the inventory segment. The root segment displayed in Figure 6-17 has four occurrences of the inventory segment beneath it. IMSADF II therefore causes the message **ENTER AMENDMENTS OR OPTION 'M' to SEE MORE DATA** to be displayed.

Since IMSADF II reserves space in the segment area (in the SPA) for only one occurrence of each segment ID, you must define aliases in order to hold multiple occurrences. The easiest way to do this is to set up the definition once in a library and employ the Rules Generator INCLUDE statement to copy the definition several times under different segment IDs. The library member consists of FIELD statements like these:

```
  FIELD     ID=ILOC,KEY=YES,LENGTH=16,POS=1,SLENGTH=10
  FIELD     SLENGTH=10,ID=PRIC,LENGTH=9,POS=21,TYPE=DEC,DEC=2
  FIELD     ID=REQC,LENGTH=7,POS=90,TYPE=DEC
  FIELD     ID=STCK,LENGTH=7,POS=114,TYPE=DEC
```

This member will be stored in a library (PDS), referenced by the ADFLIB DD name which may be added to the Rules Generator JCL procedure MFC1G.

If three segment occurrences are to be displayed, three segment aliases must be defined as follows:

```
  SEGMENT    ID=I1,PARENT=PA,NAME=STOKSTAT,LENGTH=160
  INCLUDE    MEMBER=INVTWIN
  SEGMENT    ID=I2,PARENT=PA,NAME=STOKSTAT,LENGTH=160
  INCLUDE    MEMBER=INVTWIN
  SEGMENT    ID=I3,PARENT=PA,NAME=STOKSTAT,LENGTH=160
  INCLUDE    MEMBER=INVTWIN
```

INVTWIN is the name of the member that contains the above segment definition.

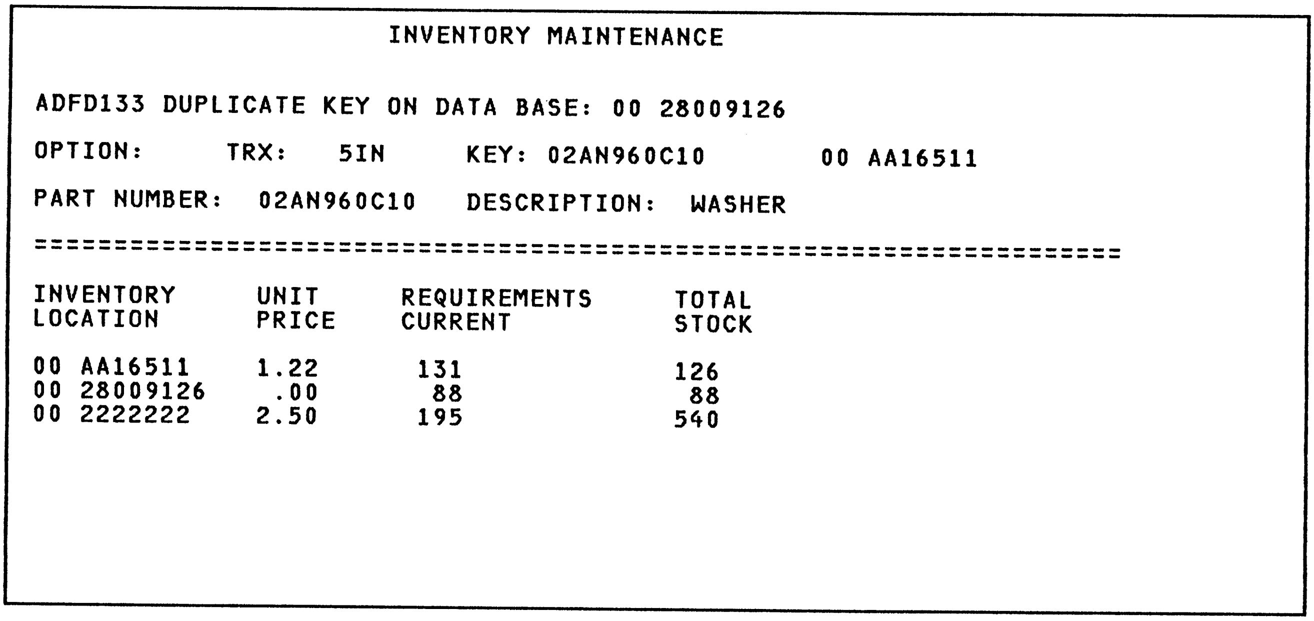
The GENERATE and screen image definitions are given below:

```
  GENERATE TRXID=IN,DBPATH=I1,TWINS=(I1,I2,I3),OPT=CVALL,
           TRXNAME='INVENTORIES',SPOS=SIMAGE
  &=1
                            'INVENTORY INFORMATION
  &=2
  &SYSMSG
  &=1
  OPTION: &OPT TRX:&TRAN KEY:&KEY
  &=1
    PART NUMBER: &5KEY.PA            DESCRIPTION: &6DESC.PA
  &=1
   ========================================================================
  &=1
    INVENTORY              UNIT             REQUIREMENTS        TOTAL
    LOCATION               PRICE            CURRENT             STOCK
   &5ILOC.I1              &5PRIC.I1        &5REQC.I1          &5STCK.I1
   &5ILOC.I2              &5PRIC.I2        &5REQC.I2          &5STCK.I2
   &5ILOC.I3              &5PRIC.I3        &5REQC.I3          &5STCK.I3
  &ENDS
```

Notice that the first twin segment appears in the DBPATH operand and is the target segment of this transaction. Its key appears as the concatenated key on the screen.

As with other data display screens, the user can amend data and cause the segments to be updated. In addition, the user can change a key. Changing the key of a DBPATH segment on a screen causes a fresh start to the transaction. Altering the key of a twin segment, however, (whether the first twin or not) causes a new segment occurrence to be inserted. In effect, this is a segment copy operation. The application developer can restrict this capability as desired by setting a mode of 6 (non-modifiable) against the key field on the screen image definition.

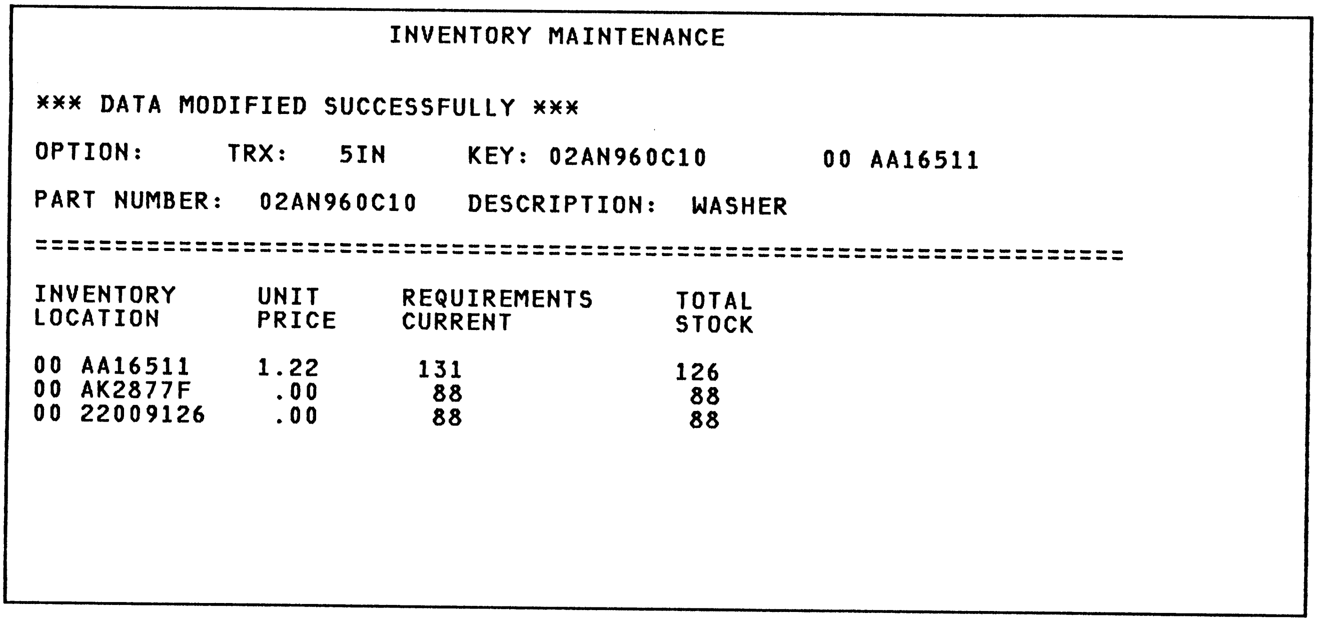
In Figure 6-18, the user has altered data on the third line, to cause the segment to be replaced, and has changed the key on the second line. Unfortunately, the new key entered duplicates one that is already on the data base.



```
                        INVENTORY MAINTENANCE

ADFD133 DUPLICATE KEY ON DATA BASE: 00 28009126

OPTION:      TRX:    5IN     KEY: 02AN960C10        00 AA16511

PART NUMBER:   02AN960C10    DESCRIPTION:   WASHER
=====================================================================

INVENTORY      UNIT      REQUIREMENTS       TOTAL
LOCATION       PRICE     CURRENT            STOCK

00 AA16511     1.22       131               126
00 28009126     .00        88                88
00 2222222     2.50       195               540
```

Figure 6-18. User Has Altered Some Data and a Key

In Figure 6-19, the user corrects the key on the second line, causing that segment to be copied and the data to be inserted under the new key.



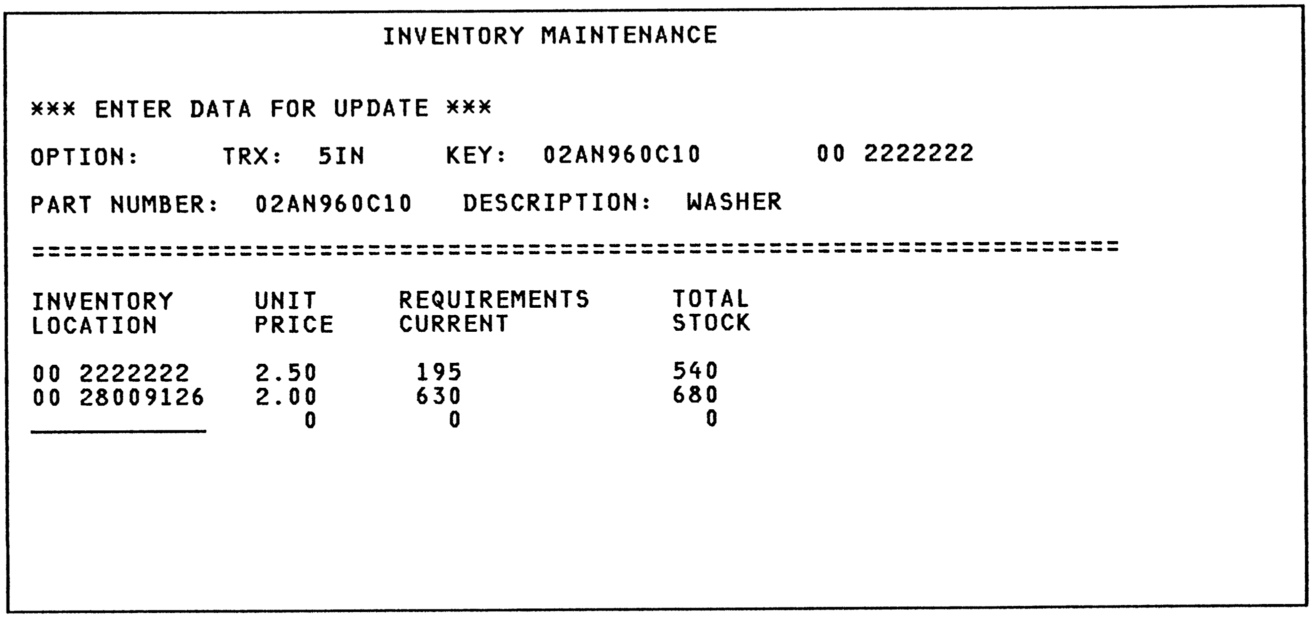
```
                        INVENTORY MAINTENANCE

*** DATA MODIFIED SUCCESSFULLY ***

OPTION:      TRX:    5IN     KEY: 02AN960C10        00 AA16511

PART NUMBER:   02AN960C10    DESCRIPTION:   WASHER
=====================================================================

INVENTORY      UNIT      REQUIREMENTS       TOTAL
LOCATION       PRICE     CURRENT            STOCK

00 AA16511     1.22       131               126
00 AK2877F      .00        88                88
00 22009126     .00        88                88
```

Figure 6-19. User has corrected the Key

The application developer can stop IMSADF II from performing this validity check on the key by coding DTWINC=NO on the transaction GENERATE statement. The check is performed, by default, by the Auditor. The check is performed, by default, by the Auditor after it has carried out all the field audits (P1 leg), before starting work on the message leg (P2). By making the checks at that point, it can detect any changes to twin keys caused by audit operations or by MAPPER calls made by exits or special processing routines. These calls cause twin segments to be inserted.

Note that in the case of non-twin segments (i.e., segments not named in the TWINS operand) a MAPPER call that changes a key will cause a segment to be inserted but a move performed via an audit operation will not.

Next the user enters M in the option field in order to receive the next page, seen in Figure 6-20. The user now has space to enter further segments. This paging process can go on as long as necessary. Option R will cause a return to the first twin segment occurrence.



```
                         INVENTORY MAINTENANCE

 *** ENTER DATA FOR UPDATE ***

 OPTION:     TRX:  5IN     KEY:  02AN960C10        00 2222222

 PART NUMBER:  02AN960C10   DESCRIPTION:  WASHER

 ======================================================================

 INVENTORY     UNIT     REQUIREMENTS      TOTAL
 LOCATION      PRICE    CURRENT           STOCK

 00 2222222    2.50      195              540
 00 28009126   2.00      630              680
 ___________      0        0                0
```

Figure 6-20. User Has Entered Request M for Next Page

TWIN PROCESSING CONTROL

Although many functions are provided as standard, statements are provided in the high level audit language to assist in developing extra logic, such as for deleting segments. Conventional subscripting through PL/I arrays or COBOL OCCURS clause items is not provided in IMSADF II but something similar is provided with the SETTWIN, SETARRAY, DOTWIN, and ENDTWIN statements.

A group of statements delimited by DOTWIN and ENDTWIN will be repeated according to the numbers specified on the DOTWIN statement. Prior to the DOTWIN, a SETTWIN statement defines the twin segments against which the repeated statements are to be executed.

For example, to retrieve two segments using the aliases I2 and I3, code:

```
  SYSID = SAMP
  SEGID = TW
  FIELD = FLAG
  PRELIM
  P2
  SETTWIN = 'I2,I3'
  DOTWIN = 1 TO 2
    IF  GN   SAPAKEY  I2  OK
    ENDIF
  ENDTWIN
```

Any reference to I2 or to a field in I2 will be interpreted as a reference to the current segment for the iteration.

If it were necessary in this application to allow the user to delete segments by entering the letter D against any line, it would be necessary to define an array in a pseudo segment. Suppose the fields DFL1, DFL2, and DFL3 are defined on the screen for this purpose. A pseudo segment would be defined.

```
SEGMENT ID=TW,TYPE=PS
FIELD ID=DFL1,BYTES=1,FAUDIT=YES,MSG=YES
FIELD ID=DFL2,BYTES=1,FAUDIT=YES,MSG=YES
FIELD ID=DFL3,BYTES=1,FAUDIT=YES,MSG=YES
```

The transaction GENERATE statement would have TSEGS=TW coded. The flags can be declared through the SETARRAY statement. For example:

```
SYSID = SAMP
SEGID = TW
FIELD = DFL1
PROCESS
P2
SETTWIN = 'I1,I2,I3'
SETARRAY = SATWDFL1
DOTWIN = 1 TO 3
   IF  DFL1 = 'D'
       IF DLET IMMED KEYFIELD I1 OK
           NOP
       ENDIF
   ENDIF
ENDTWIN
```

The maximum number of segments permitted in a SETWIN statement is 100. The DOTWIN statement can name fields that contain the numbers (must be decimal, packed or binary) and need not start at one; the range must not exceed the number of twins.

The SETARRAY statement implicitly declares an array by identifying the first field. That field must be defined on a Rules Generator FIELD statement, which must be followed by enough other FIELD statements in the same segment or pseudo segment to satisfy the range of the DOTWIN iterations.

Both the twin segments and the segment containing the array fields must be named in the TSEGS operand or the TWINS operand (or named or implied in the DBPATH operand) of the transaction GENERATE statement.

PRIMARY KEY AUDIT

Primary key audit can be used on the first segment named in the TWINS operand just as on any DBPATH segment.

The COFIELD facility can be used also to edit the key entered by the user on the primary key selection screen or into the concatenated key area on any screen. The primary key audit logic is responsible for performing this editing.

Primary key audit or COFIELD are not permitted on twin segment IDs other than the first one named in the TWINS operand value. No SETTWIN, SETARRAY or DOTWIN statements are to be used in key audit.

Where twins other than the first require editing of keys, this should be done with audit rules in the PROCESS phase.

The following example assumes that a pseudo segment has been defined as follows:

```
SEGMENT ID=TW,TYPE=PS
  FIELD ID=FLAG,LENGTH=1                TO CONTROL SK AUDIT
  FIELD ID=KEY1,LENGTH=10               COFIELD
  FIELD ID=KEY2,LENGTH=10               DISPLAYED FORM OF KEY
  FIELD ID=KEY3,LENGTH=10               DISPLAYED FORM OF KEY
```

The first twin segment I1 now has the following keyword settings.

```
SEGMENT ID=I1,PARENT=PA,NAME=STOKSTAT,LENGTH=160
FIELD   ID=ILOC,KEY=YES,LENGTH=16,NAME=STOCKEY,KAUDIT=Y,
        COFIELD=KEY1.TW,PAUDIT=YES,AUDIT=YES,FAUDIT=YES
FIELD   ID=PRIC,SLENGTH=10,LENGTH=9,POS=21,TYPE=DEC,DEC=2
FIELD   ID=REQC,LENGTH=7,POS=90,TYPE=DEC
FIELD   ID=STCK,LENGTH=7,POS=114,TYPE=DEC
```

The transaction is defined now as follows, with the pseudo segment fields on the screen image.

```
    GENERATE TRXID=IN,DBPATH=(I1),OPT=CVALL,
             TRXNAME='INVENTORIES',SPOS=SIMAGE,DLET=(I1,I2,I3),
             TWINS=(I1,I2,I3),
             TSEGS=TW
  &=1
                            'INVENTORY INFORMATION
  &=2
  &SYSMSG
  &=1
  OPTION:  &OPT TRX: &TRAN KEY: &KEY
  &=1
    PART NUMBER:  &5KEY.PA            DESCRIPTION:  &6DESC.PA
  &=1
  =====================================================================
  &=1
    INVENTORY              UNIT            REQUIREMENTS        TOTAL
    LOCATION               PRICE           CURRENT             STOCK
   &5KEY1.TW             &5PRIC.I1         &5REQC.I1         &5STCK.I1
   &5KEY2.TW             &5PRIC.I2         &5REQC.I2         &5STCK.I2
   &5KEY3.TW             &5PRIC.I3         &5REQC.I3         &5STCK.I3
  &ENDS
```

The following audit logic is responsible for moving the pseudo segment fields entered by the user into the corresponding data base key fields.

```
            KEY
            P0
  * PRIMARY KEY AUDIT MOVES COFIELD TO KEY FIELD OF FIRST TWIN
   SAI1ILOC = SATWKEY1
            PROCESS
            P1
  * AUDIT MOVES THE DISPLAYED TWIN KEYS TO THEIR DATA BASE FORMS
   SETARRAY = SATWKEY1
   SETTWIN  = 'I1,I2,I3'
   DOTWIN = 1 TO 3
     IF SATWKEY1 CHANGED = ON
       SAI1ILOC = SATWKEY1
     ENDIF
   ENDTWIN
```

SECONDARY KEY AUDIT

Just as secondary key audit can be used to limit the display of segment occurrences on the secondary key selection screen, so it can be used on the data display screen with twins. In fact, secondary key selection is not performed for twins, since that function is performed in twin retrieval.

When writing secondary key audit rules for twins, do not use SETTWIN, SETARRAY or DOTWIN. Write the rules to apply to the first twin segment ID. IMSADF II will automatically repeat the rules when retrieving subsequent segments.

The following example illustrates how to prevent one key value from being displayed and to stop retrieval at the first key beginning with 9.

```
            KEY
            P1
  * SECONDARY KEY AUDIT EXCLUDES OCCURRENCE 22332233
   IF SAI1ILOC = '22332233'
     SKSDISP = OFF
   ELSE
  *  SECONDARY KEY AUDIT STOPS RETRIEVAL WHEN A KEY STARTS WITH 9
     IF SAI1ILOC = '9'
       SKSDISP = STOP
     ENDIF
   ENDIF
```

If COFIELD is used for key editing as discussed under primary key audit, a secondary key audit routine is also needed to move the key value of the first twin back into the pseudo segment field. A preaudit is also needed to do the same for all the other twins.

The following example completes the picture for COFIELD and twins, showing the audit rules needed to move from the data base form to the displayed form in the pseudo segment.

```
          KEY
          P1
* SECONDARY KEY AUDIT EXCLUDES OCCURRENCE 22332233
 IF SAI1ILOC = '22332233'
   SKSDISP = OFF
 ELSE
*  SECONDARY KEY AUDIT STOPS RETRIEVAL WHEN A KEY STARTS WITH 9
   IF SAI1ILOC = '9'
     SKSDISP = STOP
   ELSE
*    FIRST TWIN KEY NEEDED FOR CONCATENATED KEY.
*    USE FLAG TO PREVENT OTHER TWIN KEYS OVERLAYING IT.
     IF SATWFLAG = ' '
       SATWKEY1 = SAI1ILOC
       SATWFLAG = '1'
     ENDIF
   ENDIF
 ENDIF
          PRELIM
          P1
* RESET FLAG USED IN SECONDARY KEY AUDIT
  SATWFLAG = ' '
* PREAUDIT MOVES THE TWIN KEY FIELDS TO THEIR DISPLAYED FORMS
 SETARRAY = SATWKEY1
 SETTWIN  = 'I1,I2,I3'
 DOTWIN = 1 TO 3
   IF SAI1ILOC ¬= '          '
     SATWKEY1 = SAI1ILOC
   ENDIF
 ENDTWIN
```

Notice the use of a flag field. The secondary key audit processing is performed against each twin segment. The flag is used to ensure that only the key of the first twin is moved into the COFIELD. During preaudit, this flag is reset ready for a later re-retrieval should the user enter the M or R options or cause insertions or deletions.

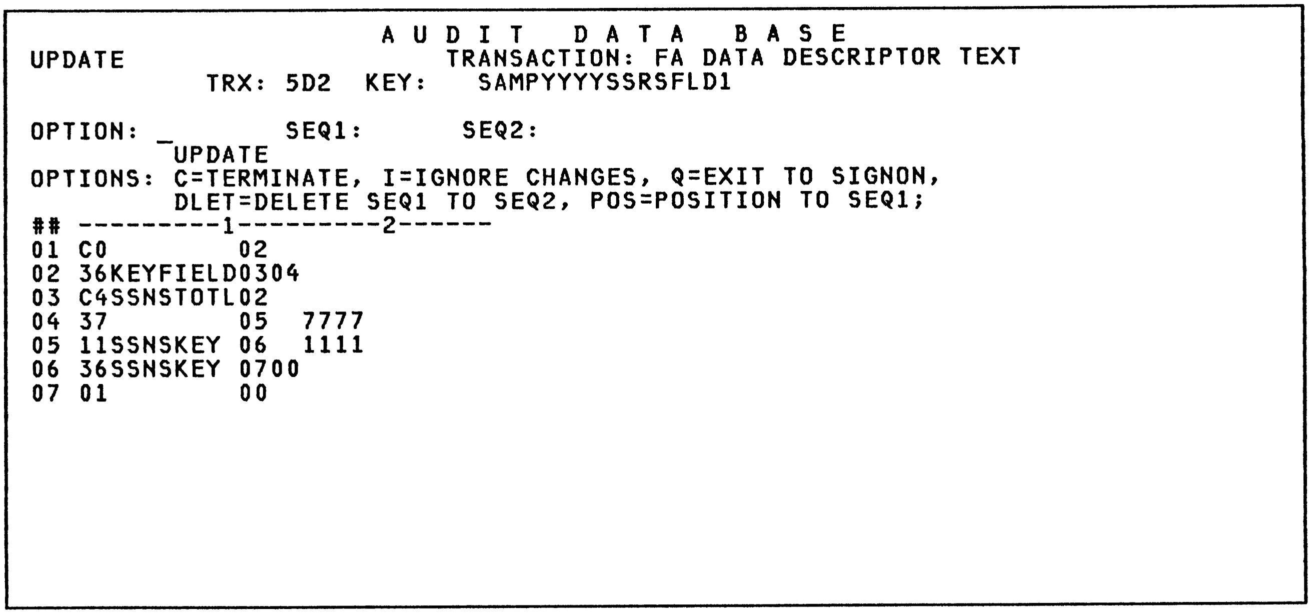
TEXT UTILITY

The text utility function exists to handle multiple segment occurrences with certain restrictions:

- Segments must be dependent and have a textual format
- Segment key fields are no more than 20 bytes
- A single data field does not exceed 77 bytes
- Total segment length including key and data must be 78 bytes or less

Several segments in the IMSADF II dynamic rules data bases are of this kind. An example is the audit data descriptor segment.

Figure 6-21 shows the screen. The user reaches it by entering OPTION D and TRANSACTION MODE 5 on the Primary Option Menu. He will be prompted for a transaction ID and a parent key by the Secondary Option Menu and by key selection if he does not enter them on the Primary Option Menu.



```
                      A U D I T   D A T A   B A S E
UPDATE                    TRANSACTION: FA DATA DESCRIPTOR TEXT
           TRX: 5D2  KEY:   SAMPYYYYSSRSFLD1

OPTION: _       SEQ1:      SEQ2:
         UPDATE
OPTIONS: C=TERMINATE, I=IGNORE CHANGES, Q=EXIT TO SIGNON,
         DLET=DELETE SEQ1 TO SEQ2, POS=POSITION TO SEQ1;
## ---------1---------2------
01 C0        02
02 36KEYFIELD0304
03 C4SSNSTOTL02
04 37        05  7777
05 11SSNSKEY 06  1111
06 36SSNSKEY 0700
07 01        00
```

Figure 6-21. Text Utility Example Using the Audit Data Base

Here are the Rules Generator statements that built this transaction. First the operation descriptor segment is defined:

```
  SEGMENT   ID=FA,LENGTH=28,NAME=MFFAAR01,
            KEYNAME=MFFAKEYF,PARENT=GF
  FIELD     ID=SEQ#,LENGTH=2,KEY=YES,REQ=YES,DISP=YES,
            MODE=5,SNAME='SEGMENT SEQ'
  FIELD     ID=DCDE,SNAME='DESCRIPTOR CODE',LENGTH=2,REQ=YES,REL=YES
  FIELD     ID=RFLD,SNAME='RELATED FIELD',LENGTH=8
  FIELD     ID=NTRU,SNAME='NEXT TRUE SEQ NO',LENGTH=2,REL=YES
  FIELD     ID=NFLS,SNAME='NEXT FALSE SEQ NO',LENGTH=2,REL=YES
  FIELD     ID=MSG#,SNAME='MESSAGE #',LENGTH=4
```

The GENERATE statement to define a text utility transaction is similar to an ordinary transaction GENERATE statement. Here is a standard processing transaction GENERATE statement:

```
  GENERATE  TRXID=DF,TRXNAME='FIELD AUDIT DATA DESCRIPTOR',
            OPT=CVALL,DBPATH=FA
```

Here is the text utility transaction definition:

```
  GENERATE  TRXID=D2,TRXNAME='FA DATA DESCRIPTOR TEXT',
            OPT=TUALL,DBPATH=FA
```

Other GENERATE statements (OPT=CVSYS, SOM, STLE, SGALL) for text utility are the same as for standard processing, as are the IMS/VS transaction and PSB definitions.

CHAPTER 7. SECONDARY TRANSACTIONS AND IMS/VS MESSAGE ROUTING

This chapter describes secondary transactions and message routing in an IMS/VS environment. The analogous facilities in a CICS/OS/VS environment are described in "Secondary Transaction - Output Message Routing" in IMS Application Development Facility II Version 2 Release 2 Application Development Reference.

IMS/VS provides facilities for application programs to send messages to terminals, identified by logical terminal name, and to programs, identified and invoked by a transaction code. It is required that transaction codes differ from logical terminal names in the IMS/VS system definition so that the same application programming conventions can be used for both.

IMSADF II invokes the IMS/VS functions when requested to do so by appropriate rules. Messages can be sent to transaction programs written in normal COBOL, PL/I, Assembler or FORTRAN as well as to transactions implemented using IMSADF II. The receiving, or secondary, transactions can be batch message processing programs (BMPs) or nonconversational message processing programs (MPPs) but not conversational MPPs. For message switching between conversational transactions, see Chapter 6, "Complex Transactions."

If a conversational IMSADF II transaction sends a message to a secondary transaction (nonconversational), the conversation continues without the end user being aware of the secondary transaction. This is because the secondary transaction is executed asynchronously and cannot communicate with the end user's terminal as long as the conversation remains active (e.g., until the user signs off).

To request message sending to terminals (e.g., printers, displays) or to secondary transactions, three items of information must be supplied:

- The format and content of the message.

  This is done via an Output Format Rule or via a GENERATE statement with OPT=OMFS.

- The conditions under which the message is to be sent.

  This is done by means of the STX operand on the transaction GENERATE statement or by audit rules.

- The routing of the message; that is, to which transactions or terminals it is to be sent.

  Routing information must be placed in the IMSADF II Message Data Base.

OUTPUT FORMAT RULE

The format and content of a message to be sent to a transaction (IMSADF II or not) are defined through the Rules Generator. An Output Format Rule is defined by a SEGMENT statement with TYPE=OUT. Special FIELD statement operands permit text and system information, as well as data fields, to be included in a message. The data fields are derived from fields in the transaction, in data base or pseudo segments.

Example

```
SEGMENT   ID=FS,LENGTH=38,TYPE=OUT
FIELD     TEXT='SAMPB05C 5SC',LENGTH=12
FIELD     ID=KEY,SEGID=PA,LENGTH=17
FIELD     ID=KEY,SEGID=PD,LENGTH=2,TYPE=DEC
FIELD     ID=INVC,SEGID=PD,LENGTH=1
FIELD     KWNAME=SPAMANNO
```

The full name of the Output Format Rule is of the form:

ssORxx01

where:

ss is the first two characters of the application system ID
OR is a literal
xx is the output segment ID
01 is a literal (note letter O, number 1)

As the example shows, textual portions of the message are defined with the TEXT operand of the FIELD statement. System information is included in the message by means of the KWNAME operand.

Here is a list of KWNAME operand values with their default lengths. These may be varied by use of the LENGTH operand. All are alphanumeric unless otherwise stated.

NAME	LENGTH	DESCRIPTION
SPATERM	4	Terminal no: T# = a literal and nn = bytes 7 and 8 of the logical terminal name if those bytes are numeric; otherwise nn is blank
SPAMANNO	6	User ID signed on
SPAUSER	11	Name of user signed on (from Sign-On Profile Data Base)
SPAPROJ	1	First byte of project/group signed on
SPAGROUP	1	Second byte of project/group signed on
SPATRX	3	Current transaction mode and ID
SPATRXCD	1	Current transaction mode
SPATRXSG	2	Current transaction ID
SPAKEYID	255	Concatenated key for current transaction
SPAERMSG	50	Field that contains system message displayed to end user
SPASHOTR	8	IMS/VS transaction code in progress
SPATRANS	8	IMS/VS transaction code at beginning of SPA
SPASYSID	4	Application system ID
SPADATE	5	Date user signed on (Julian date: YYDDD)
SPASIGNON	6	Time user signed on (HHMMSS)
SPALTERM	8	End user's logical terminal name
SPACGTRX	3	New transaction mode and ID requested by special processing program

In the example, the layout of the message begins with text defining an IMS/VS transaction code. This is standard IMS/VS practice when sending messages to secondary transactions, so that the receiving program sees the message in the format in which terminal messages are received. In this case, the receiving program is an IMSADF II nonconversational application, which also requires the three-character transaction mode and ID as shown. See Chapter 11, "Nonconversational Processing" for a fuller explanation of transaction codes.

The layout of the message must be defined to suit the receiving program.

Data fields in the message have the SEGID operand to indicate which segment in the sending transaction each field is coming from. In the same Rules Generator run there must be fields with the same IDs defined in data base or pseudo segments with IDs matching the value of SEGID. Thus, statements like the following will be present:

```
  SEGMENT   ID=PA,LENGTH=50,PARENT=0,NAME=PARTROOT
  FIELD     ID=KEY,LENGTH=17,KEY=YES,NAME=PARTKEY
  SEGMENT   ID=PD,LENGTH=85,PARENT=PA,NAME=STANINFO
  FIELD     ID=KEY,LENGTH=2,KEY=YES,TYPE=DEC,NAME=STANKEY
  FIELD     ID=INVC,LENGTH=1,POS=21
```

These segments must be present in the transaction that sends the message described by the output segment FS.

OUTPUT MFS

Messages sent to a terminal can be implemented using the facilities described above. However, you must write your own MFS statements if the receiving terminal requires the use of the IMS/VS Message Format Service.

The Rules Generator will generate an Output Format Rule and the required MFS source statements for sending messages to printers when a GENERATE statement is coded as follows:

```
  GENERATE  OPT=OMFS,SPOS=SIMAGE,
            ORID=xx,PRINTER=p
```

where:

xx is the output segment ID
p is the printer terminal type

The xx value will be used to refer to the message in the transaction that sends it. It should be different from the ID of any segment.

Note: Do not define a SEGMENT statement with TYPE=OUT with this segment ID; the Rules Generator will build an Output Format Rule from the GENERATE OPT=OMFS statement.

The p value can be as follows:

1	3270P	model 1	(119 characters per line)
2	3270P	model 2	(119 characters per line)
3	SCS1	printer	(131 characters per line)

If the PRINTER operand is omitted, MFS statements are generated for a display terminal. In this case, the DEVNAME and DEVTYPE operands can be coded to select the appropriate device type for formatting. The GENERATE OPT=OMFS statement must be followed by an image of the output like a screen image. Since printer lines are wider than 80 bytes, each line to be printed is represented by two lines in the image. The first 66 bytes of a printed line are represented by one line in the image and the remaining bytes (53 or 65) by the succeeding line in the image.

The screen image that defines a printed format can include fields from segments defined in this Rules Generator run. These segments must be present in the transaction that sends the message.

The image can use the tabular form. It cannot include the system fields or the KWNAME operand values listed in the previous section.

Example

The following printer image refers to fields in the PA and PD segments. Notice that the space control lines &=n must be followed by a second line because each pair of lines corresponds to one line in the image. Comment lines (beginning &*), however, are not treated in pairs.

```
GENERATE    OPT=OMFS,SPOS=SIMAGE,
            ORID=AV,PRINTER=2

PART  STATUS
&*    THE NEXT 2 LINES ARE A PAIR
&=5

PART NUMBER:  &6KEY.PA

&4

INVENTORY CODE:  &6INVC

&ENDS
```

DEFINING MESSAGE SENDING CONDITIONS

The transaction that sends the message is defined with the usual GENERATE statement but with an extra operand indicating that IMS/VS message sending is required. Here is an example:

```
GENERATE  TRXID=PM,TRXNAME='PART MAINTENANCE',
          OPT=CVALL,DBPATH=PD,
          STX=(TRX,FS,OK,4)
```

The STX operand reads: send a secondary transaction (TRX) message, of format FS, if the transaction terminates without error (OK) when adding a new data base segment (transaction mode 4).

The full definition is:

```
STX=( MFS ,xx [,OK][,ER] ,mode)
      TRX
```

where,

MFS means use an MFS Message Output Descriptor (MOD) named ssORxx01, the full name of the Output Format Rule. (Note the literals - letters OR and letter O numeral 1.) The MFS may be generated with a GENERATE OPT=OMFS statement, or you may code it in accordance with this naming convention.

TRX means do not use MFS. The message is going to a secondary transaction or a terminal that does not require MFS.

xx is the ID of the TYPE=OUT segment that defines the message format or is the value of the ORID operand on a GENERATE OPT=OMFS statement.

OK means send the message if the transaction terminates without error.

ER means send the message if the transaction terminates with an error. In standard processing the error can be an invalid DL/I status code received when attempting to update the data base or an error condition during preaudit. An Auditor error during the update phase does not count, since the user is expected to correct the error and allow the transaction to complete successfully.

mode is the transaction mode (1 to 6) in which the transaction must be used in order for the message to be sent. If, for instance, the message should be sent in modes 4 and 5, code two similar STX operands on the same GENERATE statement, one with n=4 and the other with n=5. If n=0, all the modes (1-6) are implied.

One or both of OK or ER may be coded with the mode, which is then required. When both are coded, the message will be sent whether the transaction terminates successfully or not.

CONTROLLING MESSAGE SENDING THROUGH THE AUDITOR

It is also possible to control message sending through the Auditor. In that case, the OK, ER, and mode operands values should not be present at all.

To control message sending through the Auditor, simply code the statement SEND 'ssORxx01', naming the intended Output Format Rule within the quotes. The operation code can be used in any of the three Audit data base legs during preaudit or the update phase. The function is quite separate from automatic message sending.

Messages will not normally be sent at the time the Auditor processes the SEND statement but later, under the control of the transaction driver, after data base updates have been performed. The values of any fields included in messages will be the values at the time they are sent.

If you want to send a message at the time the Auditor processes the SEND statement, it should be written with the immediate option thus:

```
SEND IMMED 'ssORxxol'.
```

If a message is to be sent using the message routing capability of IMS/VS Multiple Systems Coupling (MSC), code the DIRECT statement thus:

```
DIRECT 'ssORxxol' TO 'msclink'
```

where msclink is the name of the MSC link. The _IMS/VS Application Programming Manual_ (SH20-9026) explains the use of MSC.

Example

If the inventory code falls below 7, send an immediate secondary transaction to check stock levels and print a change in part status message.

Significant Rules Generator statement:

```
GENERATE  TRXID=PM,TRXNAME='PART MAINTENANCE',
          OPT=CVALL,DBPATH=PD,
          STX=(TRX,FS),STX=(MFS,AV)
```

High level audit language:

```
SYSID = SAMP
SEGID = PD
FIELD = INVC
IF INVC < 7
  SEND IMMED 'SAORFS01'
  SEND 'SAORAV01'
  ENDIF
```

Up to 60 secondary transaction or output terminal messages can be sent by a transaction. The STX operand can be coded many times on the GENERATE statement and many Output Format Rules may be used.

MESSAGE ROUTING

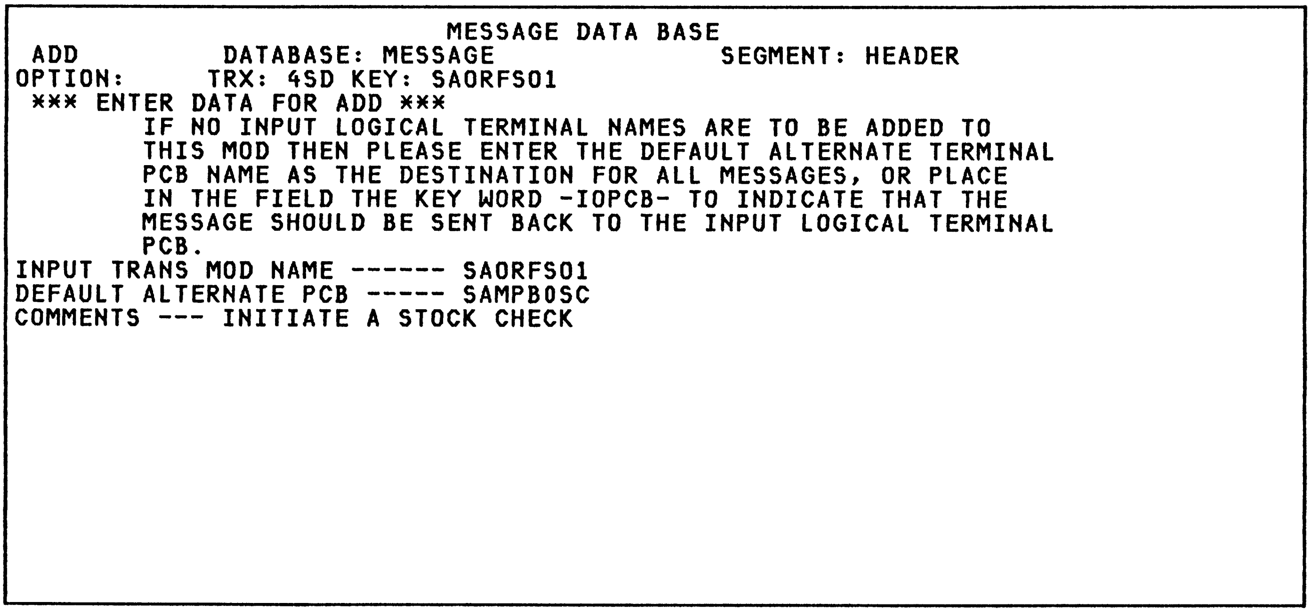
You must tell IMSADF II where to send the message if MFS is used to format a message to a terminal. This is indicated in the STX operand of the GENERATE statement. Message destinations will be either transaction codes or logical terminal names.

Routing information is stored in the Message Data Base. A routing header must be created for each Output Format Rule. Figure 7-1 shows its layout. The full name of the Output Format Rule is the key.

8	8	62
Full name of Output Format Rule ssORxx01	Destination logical terminal or secondary transaction code	Comments

Figure 7-1. Layout of Routing Information

Figure 7-2 illustrates the online transaction for creating and maintaining the routing header.



```
                                 MESSAGE DATA BASE
 ADD         DATABASE: MESSAGE              SEGMENT: HEADER
OPTION:     TRX: 4SD KEY: SAORFS01
 *** ENTER DATA FOR ADD ***
        IF NO INPUT LOGICAL TERMINAL NAMES ARE TO BE ADDED TO
        THIS MOD THEN PLEASE ENTER THE DEFAULT ALTERNATE TERMINAL
        PCB NAME AS THE DESTINATION FOR ALL MESSAGES, OR PLACE
        IN THE FIELD THE KEY WORD -IOPCB- TO INDICATE THAT THE
        MESSAGE SHOULD BE SENT BACK TO THE INPUT LOGICAL TERMINAL
        PCB.
INPUT TRANS MOD NAME ------ SAORFS01
DEFAULT ALTERNATE PCB ----- SAMPBOSC
COMMENTS --- INITIATE A STOCK CHECK
```

Figure 7-2. Creating a Routing Header

For certain message sending applications it is necessary to direct messages to different terminals depending on which terminal originated the transaction. Typically, a number of terminals at one location will require printout on a local printing terminal while terminals elsewhere wish their printout to be directed differently. To support such message routing, detail segments can be placed under the routing header in the Message Data Base (see Figure 7-3).

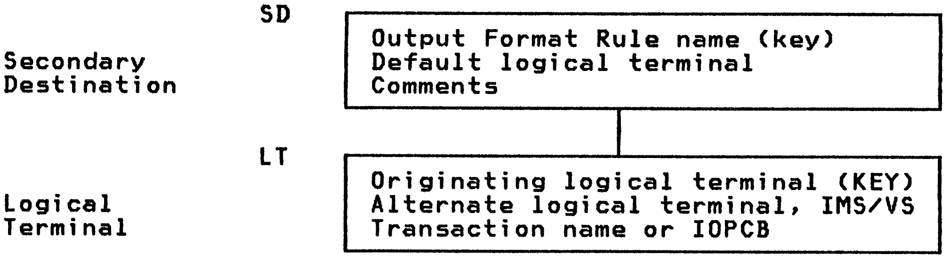
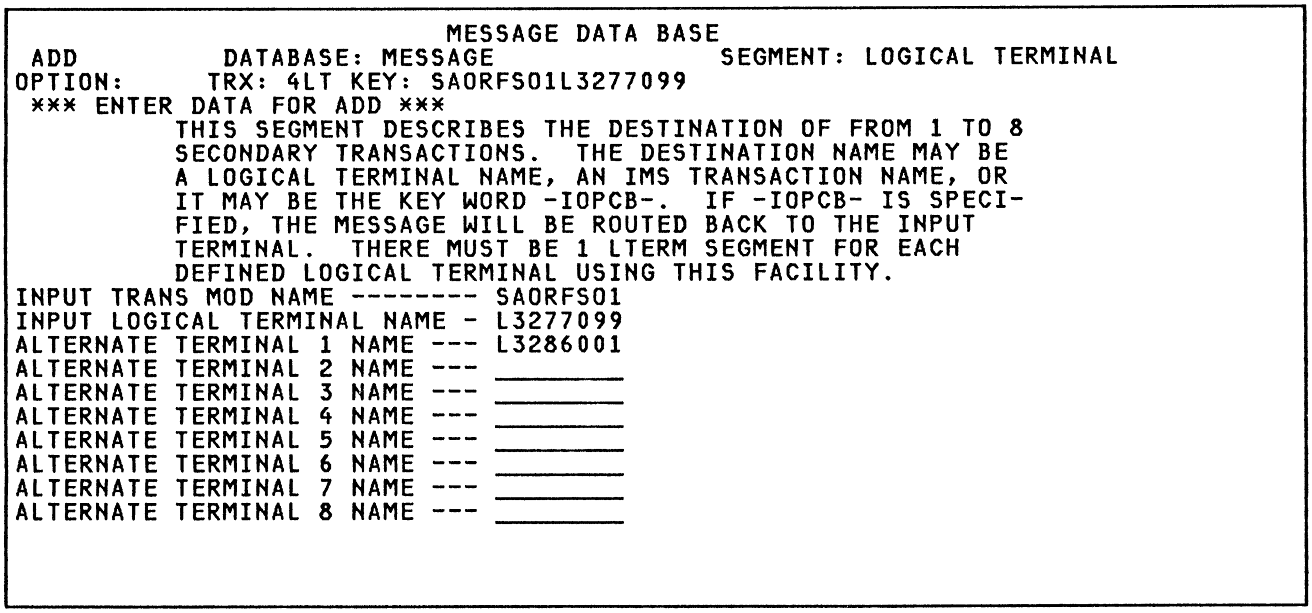

SD
Secondary Destination
Output Format Rule name (key)
Default logical terminal
Comments
LT
Logical Terminal
Originating logical terminal (KEY)
Alternate logical terminal, IMS/VS
Transaction name or IOPCB

Figure 7-3. Message Routing to Multiple Terminals



```
                          MESSAGE DATA BASE
 ADD        DATABASE: MESSAGE             SEGMENT: LOGICAL TERMINAL
OPTION:      TRX: 4LT KEY: SAORFS01L3277099
 *** ENTER DATA FOR ADD ***
          THIS SEGMENT DESCRIBES THE DESTINATION OF FROM 1 TO 8
          SECONDARY TRANSACTIONS.  THE DESTINATION NAME MAY BE
          A LOGICAL TERMINAL NAME, AN IMS TRANSACTION NAME, OR
          IT MAY BE THE KEY WORD -IOPCB-.  IF -IOPCB- IS SPECI-
          FIED, THE MESSAGE WILL BE ROUTED BACK TO THE INPUT
          TERMINAL.  THERE MUST BE 1 LTERM SEGMENT FOR EACH
          DEFINED LOGICAL TERMINAL USING THIS FACILITY.
INPUT TRANS MOD NAME -------- SAORFS01
INPUT LOGICAL TERMINAL NAME - L3277099
ALTERNATE TERMINAL 1 NAME --- L3286001
ALTERNATE TERMINAL 2 NAME --- ________
ALTERNATE TERMINAL 3 NAME --- ________
ALTERNATE TERMINAL 4 NAME --- ________
ALTERNATE TERMINAL 5 NAME --- ________
ALTERNATE TERMINAL 6 NAME --- ________
ALTERNATE TERMINAL 7 NAME --- ________
ALTERNATE TERMINAL 8 NAME --- ________
```

Figure 7-4. Creating a Message Routing Detail Segment

For each logical terminal that can originate the transaction, a detail segment is set up with the originating logical terminal name as key and with the destination logical terminal names (up to eight are allowed) as data. Figure 7-4 illustrates the LT transaction that creates and maintains the detail segments. If the system is performing message sending and finds no detail segment with key equal to the originating logical terminal name, it sends the message to the destination logical terminal name in the routing header, which is thus the default destination.

Once again, the same segment types in the Message Data Base are being used but the message text, automatic message sending, and user message header segments are kept apart from the IMS/VS message routing headers by the key value naming convention.

CHAPTER 8. SPECIAL PROCESSING

Previous chapters have described how to develop transactions by writing static and dynamic rules. Processing controlled solely by rules is known as "standard processing."

Special processing is an extension to standard processing. Special processing routines (SPRs) are written in COBOL, PL/I, or Assembler. They are executed under the control of the transaction driver, but perform functions that the transaction driver cannot do. Rules are coded to control the transaction driver in much the same way for special as for standard processing.

Special processing is used for complex, application-dependent logic for which audit rules are too cumbersome. The Auditor does not provide nested array manipulation. Therefore, coding that applies to nested groups of repeated fields or segments must be coded repetitively for each occurrence with different names.

OVERALL FLOW

The end user does not have to distinguish between standard and special processing transactions. On the Primary or Secondary Option Menu he selects a transaction ID; the system takes care of switching to the correct program if the transaction is defined as special processing on the Rules Generator GENERATE statement.

Figure 8-1 shows how the system switches to a unique transaction driver for each special processing routine. (Compare Figure 8-1 to Figure 2-1 on page 2-2.)

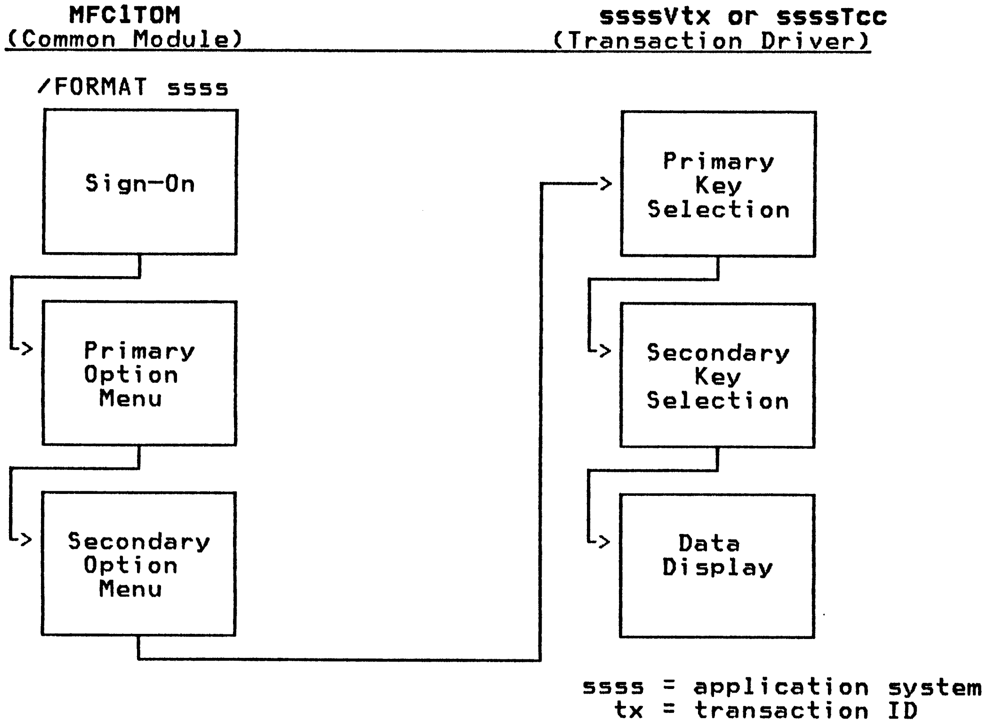


Figure 8-1. Conversational Program Flow for Special Processing

Although switching to the transaction driver is transparent to the user, the transaction itself may not behave exactly like standard processing. To begin with, key selection is optional: the program may retrieve segments directly without the user interaction provided by key selection. The program may depart from the usual convention of displaying data, receiving amendments, and performing updates. Most commonly, however, it will use all the functions of standard processing and simply perform extra computations and updates at the time the transaction driver performs its updates. Figure 8-2 shows how the program can be invoked (using the BYPASS operand of the GENERATE statement).

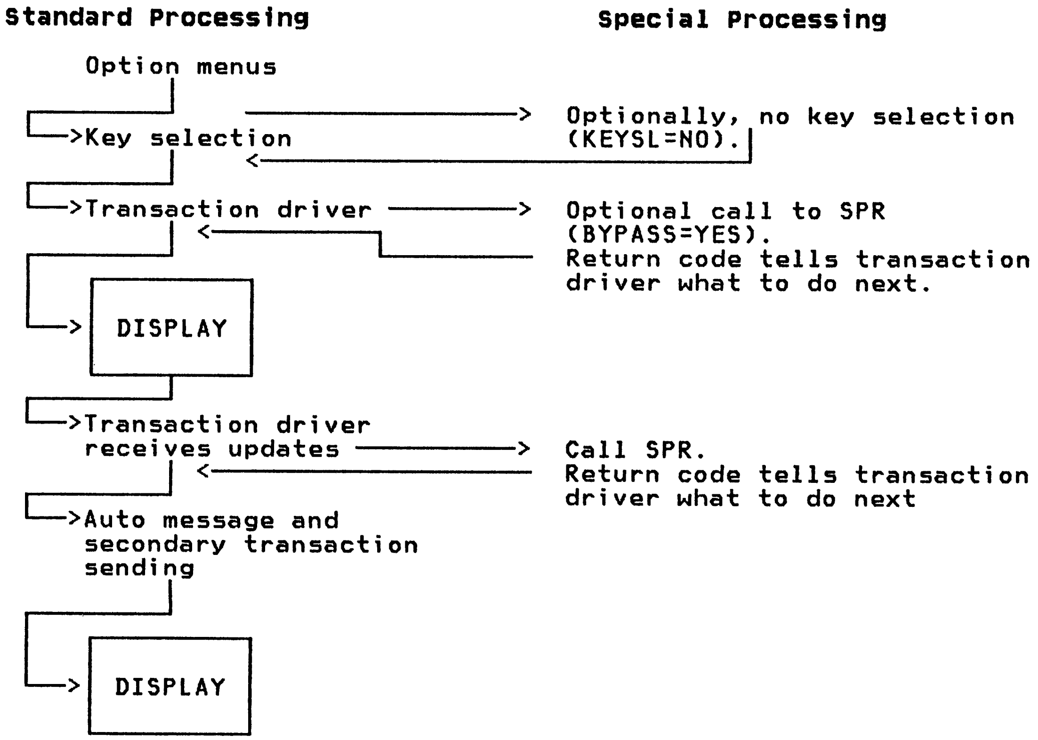

Standard Processing
Special Processing
Option menus
>Key selection
Optionally, no key selection (KEYSL=NO).
>Transaction driver
Optional call to SPR (BYPASS=YES).
Return code tells transaction driver what to do next.
DISPLAY
>Transaction driver receives updates
Call SPR.
Return code tells transaction driver what to do next
>Auto message and secondary transaction sending
DISPLAY

Figure 8-2. Standard vs. Special Processing

STATIC RULES

On the GENERATE statement for the transaction, code the following additional operands:

SPECIAL=YES   Requests special processing.

BYPASS=YES   Bypass the initial screen display and call the special processing routine (see Figure 8-2). The SPR may request the transaction driver (via a return code) to continue with screen display. Preaudit processing is performed prior to calling the SPR.

BYPASS=NO   Do not call the SPR before screen display. (This is the default.) The program will always be called during update (in transaction modes 1 to 5 - not in mode 6 which has no update phase) regardless of the BYPASS operand.

KEYSL=YES   Requests Primary and Secondary Key Selection for the DBPATH segments. The default is YES if a DBPATH is coded and NO if not.

LANG=COBOL PL/I ASMINT	Programming language in which the SPR is written. COBOL is the default. ASMINT means Assembler.

For example:

```
  GENERATE     TRXID=WI,TRXNAME='WORKING INVENTORY',
               OPT=CVALL,DBPATH=(PD,IV),
               TSEGS=(WC,WX),SPECIAL=YES,
               LANG=PL/I,BYPASS=YES,KEYSL=YES
```

This GENERATE statement could be coded for the sample problem data base introduced in Chapter 1, "IMSADF II Concepts and Overview." The special processing routine will rely on key selection to retrieve inventory and standard information segments (IV, PD, and PA) but will retrieve a work center segment (WC) from another data base under its own control before the screen is displayed. It does this itself because the key field must be derived from data in the part record through a complex calculation.

The TSEGS operand is used to reserve space in the segment area (in the SPA) for segments to be retrieved outside key selection. WX is a pseudo segment ID; these too can be used in special processing.

SCREEN FORMATTING

Just as in standard processing there is one screen layout for each transaction ID. The screen can have the default layout (SPOS=AUTO) or use screen image (SPOS=SIMAGE).

There is one difference in default screens for special processing and it concerns the default modes of fields in data base segments (DBPATH or TSEGS). If MODE is not specified on the SEGMENT or FIELD statement, it is assumed to be 6 (nonmodifiable) for all fields. For standard processing, the default mode is 6 for keys and 5 (modifiable) for others. Pseudo segment fields still default to mode 5.

The reason for the difference is that special processing routines do not always update all the data base segments displayed; updates are under program control. The user should be allowed to amend data base fields on the screen only if the corresponding data base updates will be performed.

PROGRAM CALLS

A special processing routine is entitled to use the services available to the transaction driver itself. Normally, it does not perform direct DL/I or DB2 calls although that is possible. It can rely on key selection to retrieve segments specified in DBPATH. It can also request the transaction driver to update all segments or tables (DBPATH and TSEGS) that are changed or marked as changed. Using these services, it is easy to write a program that performs the functions of standard processing. Such a program is given later. For most applications it is advantageous to begin with a copy of this base program and write enhancements to it rather than set out from scratch.

The following is a subset of the call services available to a special processing routine. For a full list, refer to the IMS Application Development Facility II Version 2 Release 2 Application Development Reference.

SEGUPDTE Requests the transaction driver to perform the data base updates as if in standard processing, replacing changed segments or tables (including those changed by the SPR), inserting and deleting according to the rules and taking auditing into account.

SETFLAG Controls the operation of the SEGUPDTE routine or the transaction driver by setting the segment changed, retrieved, and delete flags.

AUDITOR Auditing will be done during the update phase only if the SPR calls for it. On the phase prior to display (BYPASS=YES), preaudit will be performed after the transaction driver has loaded the DBPATH segments but before the SPR is invoked. Therefore, the SPR should not call the Auditor in the preaudit phase.

MAPPER When the program needs to access data in segments, it calls the Mapper. The selection of fields required by the program is defined in a "mapping segment." Thus, the program only receives or sends those fields with which it is concerned and is independent of the layout of the segments. It is not even concerned with the hierarchical relationships and keys of the segments if key selection and the SEGUPDTE call are used.

SEGHNDLR If the data base retrievals and updates performed by the transaction driver and DL/I Auditor calls are not enough, the special processing routine can issue DL/I calls through the IMSADF II segment handler. These calls are at the same level as DL/I Auditor calls; keys are passed in the call but the program need not handle DL/I segment search arguments (SSAs). Several subroutine calls are available, however, to allow the SPR to manipulate keys and SSAs if necessary.

SQLHNDLR This call allows DB2 tables to be processed in the same manner the SEGHNDLR call above processes DL/I segments. The call references precoded static SQL functions in a Table Handler Rule.

DISPLAY The SPR can write to display screens and printer terminals using the DISPLAYA, DISPLAYL, DISPLAYE and DISPLAYP calls. This is in addition to sending secondary transactions.

Each call sets a return code, which must be checked by the special processing routine. The return code is placed in a communication field SPARTNCD. This is one of several fields wherein the program communicates with the transaction driver or subroutines. These fields are declared in a COPY or INCLUDE member which is provided with the product and is to be compiled into each SPR. This member defines the SPA (scratch pad area) which is used as a general communication and work area by IMSADF II. It is passed to the SPR on entry as a parameter by the transaction driver. See "Program Linkage" on page 8-21.

RETURN CODE CONVENTIONS

The SPR sets a return code to the transaction driver using the Operation System Convention. For example, to set a return code of 8:

```
In COBOL:        MOVE 8 TO RETURN-CODE.
                 GOBACK.

In PL/I:         CALL PLIRETC(8);
                 RETURN;

In Assembler:    LA   15,8
                 BR   14
```

AUDITOR CALL

```
In COBOL:    CALL 'AUDITOR'.
In PL/I:     CALL AUDITOR;
```

All AFA, field auditing, and message leg rules (P0, P1, and P2) will be executed for all fields appropriately marked, just as in standard processing. The SPR should check the return code (SPARTNCD). If audit errors occur, a return code of 8 or more will be set. In that case, the SPR should return to the transaction driver, setting a return code of 8, which will cause the error message display to proceed as in standard processing. The fields in error will be highlighted and the user may enter E to view the messages. When he corrects the errors, the transaction driver will call the program again, and it should call the Auditor. This loop can be repeated as often as necessary.

See "Return Codes" on page 8-24 for a complete list of return codes that the SPR can set. The codes that can be returned in SPARTNCD from the Auditor are:

Code Meaning

0 All audits successful. No messages to send.

4 All audits successful. Messages exist for the transaction driver to send. This means that the Auditor has set a flag that will cause the transaction driver to perform automatic message sending when control is returned to it.

5 TRX mode or TRXID changed.

7 Warning messages. The Auditor has found one or more warning messages and no errors. The SPR should return control to the transaction driver with return code 8 to display the warning messages. If the user enters **U** on the message screen, the SPR is recalled with 99 in SPARTNCD. The SPR can then perform the appropriate updates. If the user does not enter U, the segment display screen is redisplayed for additional modification.

8 Data failed audit. The Auditor has found an error in the validation of one or more fields. The SPR should return control to the transaction driver at once with return code 8 to display the error notification messages.

12 Audit descriptor not found. The field was marked for auditing, but the expected audit rule was not present. The SPR should return control to the transaction driver at once with return code 8 to display the error notification and messages.

16 Field not found during automatic field assignment. A field specified for automatic field assignment in the Input Transaction Rule was not present in a Segment Layout Rule currently in the SPA. Correct the discrepancy between the rules.

SEGUPDTE CALL

```
In COBOL:     CALL 'SEGUPDTE'.
In PL/I:      CALL SEGUPDTE;
```

Segments or tables changed or added by the user at the terminal, by the Auditor, or by the SPR using the Mapper are replaced or inserted. Segments or tables marked for insertion, deletion or replacement by DL/I Auditor calls are inserted, deleted, or replaced, respectively. If the DATACOMP operand has been coded on the GENERATE statement for the transaction, the segments named in that operand value will first be retrieved and compared with their original values saved in the SPA prior to display. If any of them are unequal, no data base updates will be performed.

If the SEGUPDTE subroutine encounters an invalid DL/I or DB2 status code while performing updates, and if it has already updated some data successfully (a partial update situation), it issues a DL/I ROLL call. This will undo the updates performed by SEGUPDTE and by the SPR in the current execution; a message is sent to the user's terminal and he must sign on and start again. Return is not made to the SPR in this case.

The codes that can be returned in SPARTNCD from the SEGUPDTE call are:

Code	Meaning
-4	Segment/Table Handler Rule not found in Batch Driver Rule.
0	Successful completion.
4	Nonblank status code returned from a DL/I call but no updates yet performed. The SPR should return control to the transaction driver at once with return code 28 to display the error message.
12	Data compare (DATACOMP) failure in conversational processing.
16	No segments or tables have been changed or marked for deletion. Therefore, no updates have been performed.

SETFLAG CALL

Deleting and inserting segments with the SEGUPDTE call takes place under the control of rules defining insert and delete eligibility and DL/I Auditor calls. These rules are explained in Chapter 6, "Complex Transactions." It is also possible for the SPR to control this activity by setting flags. A SETFLAG call is provided for the purpose and should be issued before the SEGUPDTE call.

Most commonly, the SETFLAG call will be used to mark a segment for deletion. The call then takes the following form:

```
  In COBOL:  CALL 'SETFLAG' USING ID.
  In PL/I:   CALL SETFLAG (ID);
```

where ID is the two-character segment ID to be marked for deletion.

The SETFLAG call is:

```
  In COBOL:  CALL 'SETFLAG' USING ID, FLAG, SETTING.
  In PL/I:   CALL SETFLAG (ID, FLAG, SETTING);
```

where:

ID is a two-character segment ID to have indicator set.

FLAG is one character that defines which indicator is to be set:

D = delete flag (the default)
R = retrieved flag

SETTING is one character that defines how the indicator should be set:

0 = off (the default with R)
1 = on (the default with D)

Return codes are:

Code	Meaning
0	Successful completion.
4	A dependent of the flagged segment has been changed but will be deleted if SEGUPDTE is called without turning off the delete indicator.
8	Segment is not eligible for deletion.

Along with the delete and retrieved indicators, a "segment changed" indicator is used to determine how the data base segments are to be updated.

The DL/I functions performed, based on the corresponding flag settings, are:

DL/I Function	**Flags**
Delete	delete flag on + retrieved flag on
Insert	changed flag on + retrieved flag off
Replace	changed flag on + retrieved flag on

SETFLAG must be used to cause deletion of segments. The GENERATE statement DLET operand is used to define which segments are eligible for deletion. SETFLAG also must be used when an existing segment is mapped into the segment area after having been retrieved directly into the user's program area. This is necessary because the Data Mapper will turn off the retrieved flag if a key field is changed in order to set up for an insert. This technique will not work on segments specified in the DATACOMP operand of the GENERATE statement. In order to save the original data for comparison, the segment must be retrieved into the SPA (via DBPATH or TSEGS).

Figure 8-3 summarizes when these indicators are set on or off:

Indicator	**Setting**	**Condition**
changed flag	ON	• Auditing (value moved to a field) • Input from Data Display screen • Mapping if field is changed
changed flag	OFF	• Successful DL/I call via SEGHNDLR or SEGUPDTE
delete flag	ON	• SETFLAG routine • Delete call from the Auditor
delete flag	OFF	• SETFLAG routine • Successful DL/I delete call via SEGHNDLR or SEGUPDTE
retrieved flag	ON	• If segment initially loaded by the transaction driver • Successful retrieval into segment area or insert from segment area via Auditor, SEGHNDLR or SEGUPDTE • SETFLAG routine
retrieved flag	OFF	• Successful DL/I delete call via Auditor, SEGHNDLR or SEGUPDTE • Key changed when mapping into segment area • SETFLAG routine

Figure 8-3. SETFLAG Indicators

MAPPER CALL

The Mapper's purpose is to insulate the special processing routine from the actual layout of fields and segments in the segment area (within the SPA). To this end, one or more views of the data can be defined. These views are known as mapping segments and consist of fields selected from those in the segment area. When the routine requires access to data, it calls the Mapper, quoting the identifier of a mapping segment. As shown in Figure 8-4, individual fields will then be moved by the Mapper into a working storage area defined in the routine and passed as a parameter in the Mapper call.

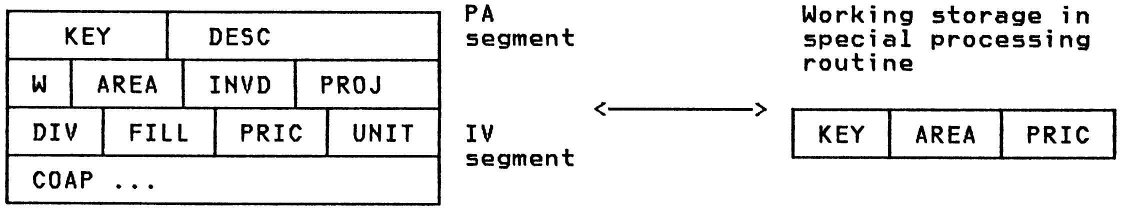


Figure 8-4. Mapping

The mapping segment is defined using the Rules Generator.

For example:

```
SEGMENT    ID=MM,TYPE=MAP
 FIELD      ID=KEY,SEGID=PA,LENGTH=17
 FIELD      ID=AREA,SEGID=IV,LENGTH=1
 FIELD      ID=PRIC,SEGID=IV,LENGTH=5,TYPE=PD,
            DEC=2,RDONLY=YES
```

This definition must appear in the same Rules Generator run as the definitions of the PA and IV segments themselves. The RDONLY operand is used to restrict the routine's ability to amend fields. If the routine calls the Mapper and requests that this mapping segment be moved from working storage to the segment area, the price (PRIC) field will not be moved. It can, however, be mapped from the segment area to working storage.

The data declarations required within the routine are as follows:

In COBOL:

```
WORKING-STORAGE SECTION.
     77 MAPID PICTURE XX VALUE 'MM'
     77 TO-WORKAREA PICTURE S9(9) COMP VALUE 0.
     77 FROM-WORKAREA PICTURE S9(9) COMP VALUE 1.
     01 MAPAREA
       03 PART-NUMBER PICTURE X(17).
       03 AREA PICTURE X.
       03 PRICE PICTURE S9(7)V99 COMP-3.
```

In PL/I:

```
DCL (TO_WORKAREA INIT(0),FROM_WORKAREA INIT(1)) BIN FIXED(31),
       01 MAPAREA,
         03 PART_NUMBER CHAR(17),
         03 AREA CHAR(1),
         03 PRICE DEC (9,2);
```

The calls to move to working storage will be:

```
In COBOL: CALL 'MAPPER' USING MAPID, MAPAREA, TO-WORKAREA.
In PL/I:  CALL MAPPER ('MM', MAPAREA, TO_WORKAREA);
```

As the example illustrates, the data declarations in the SPR must be consistent with the layout of the mapping segments. Conversion will be performed between a mapping segment and the fields in the segment area if the data types differ. The fields can be in data base or pseudo segments. The two-character ID is enough to identify the mapping segment in the call. The MAPPER subroutine is able to locate the rule, which can be link-edited with the SPR (see "Program Linkage" on page 8-21) or will otherwise be loaded from the static rules library at the time of the call. Mapping segment IDs must be unique with the application system and must differ from data base and pseudo segment IDs.

The Mapper sets a return code in SPARTNCD.

Code	Meaning
0	Successful completion.
4	Conversion error. The SPR should return to the transaction driver at once with return code 8 to display the error notification and message.

COPYSEG CALL

A function similar to MAPPER is available to simplify coding when a whole data base segment or pseudo segment must be copied into or out of a special processing routine work area. In that case, no mapping segment need be defined. Simply code a COPYSEG call quoting the ID of the segment to be copied.

Using coding similar to that for the MAPPER call, copy the IV segment into a working storage area named WORKA as follows:

```
  In COBOL:  CALL 'COPYSEG' USING SEGID, WORKA, TO-WORKAREA.
  In PL/I:   CALL COPYSEG ('IV',WORKA,TO_WORKAREA);
```

In COBOL, SEGID would be in the WORKING-STORAGE SECTION as:

```
            77 SEGID PICTURE XX VALUE 'IV'.
```

CONTROLLING COLOR AND EXTENDED HIGHLIGHTING

To alter the color of a field dynamically, issue a SETCOLOR call. To alter an extended attribute, use SETXHILT. The forms of these calls are as follows:

```
  In COBOL:  CALL 'SETCOLOR' USING FIELDID, RED.
             CALL 'SETXHILT' USING FIELDID, BLINK.

  In PL/I:   CALL SETCOLOR ('SAPDMKDP','RED');
             CALL SETXHILT ('SAPDMKDP','BLINK');
```

For COBOL, the following data declarations are needed:

```
       WORKING-STORAGE SECTION
           77  RED PICTURE X(3) VALUE 'RED'.
           77  BLUE PICTURE X(4) VALUE 'BLUE'.
           etc.
           77  BLINK PICTURE X(5) VALUE 'BLINK'.
           77  UNDERSCORE PICTURE X(10) VALUE 'UNDERSCORE'.
           etc.
           77  FIELDID PICTURE X(8) VALUE 'SAPDMKDP'.
```

The first parameter is the eight-character name of the field to be set. The second parameter is the color or attribute. Possible colors are RED, BLUE, GREEN, RED, YELLOW, TURQUOISE, and WHITE. Possible attributes are BLINK, UNDERSCORE, REVERSE, and DEFAULT.

DISPLAY CALLS

The SPR can send a message to either the input logical terminal or an alternate logical terminal when operating in conversational or nonconversational mode. This message can be up to 20 lines per logical page. The Terminal Message Writer module will unblock the message at 79 characters per line. Any word that will not fit on a line will be moved to the next line. This implies that the last (79th) character on each line must be a blank. Multiple calls can be made to the module during one output sequence.

The call sequences for the Terminal Message Writer are:

In COBOL:

```
  CALL 'DISPLAYL' USING MSGAREA, HDR, OPTION.
  CALL 'DISPLAYA' USING MSGAREA, HDR, OPTION, LTERM.
  CALL 'DISPLAYE' USING MSGAREA, HDR, OPTION.
```

In PL/I:

```
  CALL DISPLAYL (MSGAREA,HDR,OPTION);
  CALL DISPLAYA (MSGAREA,HDR,OPTION,LTERM);
  CALL DISPLAYE (MSGAREA,HDR,OPTION);
```

where:

DISPLAYL Sends the message to the input logical terminal (using the IOPCB).

DISPLAYA Sends the message to the logical terminal named in LTERM.

DISPLAYE Sends the message via the express terminal PCB. The output destination will be the input logical terminal for conversational and response nonconversational transactions. For nonresponse, nonconversational transactions, the output destination will be the printer defined in the second alternate PCB.

MSGAREA Names the area in which the current portion of the message resides. This area has two fields: a halfword containing the length of the message text followed by the message text.

HDR Names a 60-character area containing a message header to be displayed with this portion of the text.

OPTION Is a halfword that specifies the type of call currently being made.

1 = First part of message plus header. Option is set to 0 after this call.

2 = Last call with remainder of message text.

3 = Last call without additional message text.

OTHER = Add message text to the output previously received.

LTERM Is the logical terminal name of the device to receive the output.

If the message generated is to be displayed to the transaction user (through the IOPCB), a return code of 12 or 32 (conversational mode only) should be returned to the transaction driver. The terminal user can read the screen being displayed, then press the PA1 key to display successive screens. Under conversational processing, the terminal user may then redisplay the original screen by pressing ENTER.

The Terminal Message Writer returns the following codes in SPARTNCD:

Code	Meaning
0	Successful completion.
4	Nonblank status from IMS/VS when sending message. Status is in SPADLIST.
12	3rd input parm does not have last parm indicator on. Check format of CALL.

DISPLAYP CALL

The SPR can send a message to a 3284/3286 printer when operating in either conversational or nonconversational mode. This message can contain a variable number of lines per page. The Terminal Message Writer module will unblock the message at 119 characters per line. Multiple calls can be made to the module during one output sequence.

The call sequence is:

```
  In COBOL: CALL 'DISPLAYP' USING MSGAREA, HDR, OPTION, LTERM.
  In PL/I:  CALL DISPLAYP (MSGAREA,HDR,OPTION,LTERM);
```

where:

MSGAREA Names the area in which the current portion of the message resides. The format of this area is a halfword containing the length of the message text followed by the message text.

HDR Names a 60-character area containing a message header to be displayed for this message. The header line contains the 60-character message header, a sequential page number, and the last two characters of the entering LTERM. If the message header area contains blanks, no header line will be printed.

OPTION Is a halfword that specifies the type of call currently being made. Possible values are:

1 = First part of message plus header. OPTION is set to 0 after this call.

2 = Last call with remainder of message text.

3 = Last call without additional message text.

4 = Message text followed by a new page.

5 = New page.

OTHER = Add message text to the output previously received.

LTERM Is the eight-character printer LTERM name of the device to receive the output.

Each new page designation (OPTIONS 4 and 5) will cause the remaining text to be printed. Four lines with an asterisk (*) in column 1 will then be printed, followed by the header. The new page option does not contain carriage control information to physically skip the printer to a new page. If the header area is blank, it will not be printed. This allows the SPR control over the length of an output page.

Return codes from this routine are:

Code Meaning

0 Successful completion.

4 Nonblank status from IMS/VS when sending message. Status is in SPADLIST.

12 4th input parm does not have last parm indicator on. Check format of CALL.

16 Nonblank status from IMS/VS on CHNG. Status is in SPADLIST: LTERM name is probably invalid.

DIRECT CONTROL OF DATA BASE I/O

Data base retrievals can be performed by the transaction driver for the DBPATH segments in a transaction. DL/I Auditor calls can also retrieve segments identified in the TSEGS operand of the GENERATE statement. In addition, the special processing routine can retrieve segments using calls to SEGHNDLR. These can work like DL/I Auditor calls when the segments are identified in the TSEGS operand; alternatively, the SPR can request that its own defined I/O areas be used.

Data base updates will be performed by the SEGUPDTE call. The SPR can control insertions and deletions through the SETFLAG call. DL/I Auditor calls that request updates merely set flags in the same way. The special processing routine is also allowed to request immediate data base updates by calling SEGHNDLR.

SIMPLE SEGHNDLR CALLS

IMSADF II maintains the concatenated key of each DBPATH and TSEGS segment in the segment area. The segments retrieved by the transaction driver or by DL/I Auditor calls will have their concatenated keys already set up; the SEGHNDLR call need not specify a key. If the SPR is retrieving a segment for the first time, however, it must supply a fully concatenated key, although for the unqualified calls (GUU, GN and ISRT) only the parent portion of the concatenated key is used.

The simple call format is as follows:

```
  In COBOL:   CALL 'SEGHNDLR' USING ID, FUNC, KEY, OPER.
  In PL/I:    CALL SEGHNDLR (ID,FUNC,KEY,OPER);
```

where:

ID is the two-character ID of the segment to be processed.

FUNC is the DL/I or IMSADF II function code. The main ones are GU, GUU, GNQ, GN, ISRT, DLET, REPL.

KEY is the key of the segment. Can be omitted if the OPER parameter is also omitted. If this parameter is present, the field in the program containing the key value will be updated with the key of the actual segment retrieved.

OPER is a two-character code indicating how the KEY parameter is to be used. The first character can have two values:

F the KEY parameter contains the fully concatenated key of the segment.

P the KEY parameter contains only the segment key without its parents' keys. Cannot be used in the first DL/I operation against this segment.

The second character can have the following two values:

E find a segment with key equal to that specified.

G find a segment with key greater than that specified but still under the parent with key equal to that specified in the concatenated key.

If this parameter is omitted, it defaults to FE.

The DL/I function codes are as follows:

Code	**Meaning**	**Function**
GU	Get Unique	Uses the entire concatenated key to retrieve the segment with key equal to the one specified and under parents with keys equal to those specified. If two segments have the same key value, this function always retrieves the first.
GUU	Get Unique Unqualified	Uses only the parent portion of the concatenated key to retrieve the first segment occurrence under parents with keys equal to those specified.
GN	Get Next	Use after a GU or GUU call to retrieve the next occurrence of the same segment type. The call uses only the parent portion of the concatenated key, which will normally be the same as that used in a previous successful GU or GUU call. The system will only move forward in the data base to satisfy the call but will never go beyond the parent having the specified key. The GN function only works within one execution of the transaction. If the SPR finishes, the screen is displayed, the user enters amendments and the SPR is called again, the program must re-establish the data base position with a GU call before a GN is issued. IMSADF II retains the concatenated key across such steps in a conversion, but DL/I loses its position in the data base.
GNQ	Get Next Qualified	Uses the entire concatenated key to retrieve a segment with key equal to the one specified under parents with keys equal to those specified. Like GN, it moves forward in the data base and should be preceded by a GU or GUU call. It is a way of retrieving segment occurrences that have the same (nonunique) keys.
ISRT	Insert	Uses only the parent portion of the concatenated key to insert the segment under parents with keys equal to those specified. The segment being inserted contains its own key. If it is a concatenated segment, it will contain the logical parent's key twice and they must be equal (DL/I requirement). The segment is inserted immediately.
DLET	Delete	Must be preceded immediately by a Hold version of a Get call (GHU, GHUU, GHN or GHNQ). The DLET call itself does not use the concatenated key but the Get Hold call does, in exactly the same way as the Get calls without Hold. The segment is deleted immediately and is not checked for delete eligibility.

Code	Meaning	Function
REPL	Replace	Must be preceded immediately by a Get Hold call, just like DLET. The segment is replaced in the data base right away.
HDEL	(GHU+DLET)	Performs both a GHU and a DLET call but does not give the SPR the chance to look at the re-retrieved segment.
HREP	(GHU+REPL)	Performs both a GHU and a REPL call but does not let the SPR have the re-retrieved segment.

The DATACOMP operand has no effect on segments updated through the SEGHNDLR call. The program is now responsible for checking the content of a re-retrieved segment for possible outside interference before updating it.

The SEGHNDLR call sets the following return codes in SPARTNCD:

Code	Meaning
0	DL/I call successfully completed.
4	A nonblank status code has been returned from DL/I and may be found in the SPADLIST field of the SPA communication area.

"DL/I Status Codes" on page 6-23 lists the most common DL/I status codes. If the status is unexpected, the program should return to the transaction driver at once with return code 28 to display an error message.

KEY MANIPULATION SUBROUTINES

Subroutines are provided to manipulate the concatenated keys retained in the segment area. Using these subroutines can simplify the actual SEGHNDLR calls. If the concatenated key is set up with a SETKEY call, the SEGHNDLR call need not have the KEY parameter.

The four subroutines are:

GETKEY Moves either the fully concatenated key or the partial key (i.e., the key of the segment without its parents) into a program-supplied work area.

SETKEY Sets either the fully concatenated key or the partial key in the segment area (in the SPA)

ZEROKEY Sets the full or partial key in the segment area to binary zeros.

GETKINFO Retrieves three or four full words of information:

- OFFSET in concatenated key to key of this segment
- LENGTH of the key of this segment
- PCB# from the Rules Generator SEGMENT statement
- PCBADDR - the optional parm to receive the PCB address

The call formats to invoke these routines are:

In COBOL:

```
CALL 'GETKEY' USING ID, KEYA, F.
                              P

CALL 'SETKEY' USING ID, KEYA, F.
                              P

CALL 'ZEROKEY' USING ID, F.
                         P

CALL 'GETKINFO' USING ID, OFFSET, LENGTH, PCB#[, PCBADDR].
```

In PL/I:

```
CALL GETKEY (ID,KEYA,F);
                     P

CALL SETKEY (ID,KEYA,F);
                     P

CALL ZEROKEY (ID,F);
                 P

CALL GETKINFO (ID,OFFSET,LENGTH,PCB#[,PCBADDR]);
```

where:

ID is the two-character segment ID

KEYA is the area in the user's program to or from where the key is to be moved

F is a full key (default if not supplied)

P is a partial key

OFFSET, LENGTH, PCB#, and PCBADDR are the same as defined in the discussion of entry points above.

There are four possible return codes:

Code	**Meaning**
0	Successful completion of routine.
4	Segment not found or key-only segment in SPA on a GETKEY, SETKEY, or ZEROKEY call. Key-only segment on a GETKINFO call. A key-only segment is a segment in the Input Transaction Rule to maintain hierarchical information about the concatenated key. It has no segment area in the SPA, no displayable fields, and no auditing. It exists because it is a parent of a segment specified in DBPATH or TSEGS.
8	Only OFFSET and LENGTH returned on a GETKINFO call. (The segment is not in the SPA, but a Segment Handler Rule is available for the user to perform data base calls into or out of a segment area defined in the user program - the user must supply all six parameters on a SEGHNDLR call - see below.)
12	Segment is not a data base segment.

ADVANCED DATA BASE I/O

To use the facilities described in this section, you must be familiar with DL/I as described in the IMS/VS Application Programmer's Reference Manual.

Five calls are provided to allow the SPR to amend or replace the SSAs supplied by IMSADF II. These extended data base setup routines are described below.

SETCC Set up IMS/VS data base command codes on each data base level.

SETPATH Set up a path call (retrieve/update more than one segment in a single call) on segments defined in the SPA segment area. The I/O area may be in the user's SPR but the segments must be defined in the INTR.

SETSSA Set up user segment search arguments.

SETUNQ Unqualify some number of data base levels.

RSETSEGH Reset the effect of any previous setup calls.

For all calls except SETSSA, the specified segment ID must have a Segment Handler Rule. For SETSSA, if the specified segment ID is defined in the Input Transaction Rule (via DBPATH or TSEGS), I/O is allowed to/from the segment area as long as path call command codes are not present in the SSAs.

SETCC, SETPATH, and SETUNQ can be issued against the same segment ID and will have a cumulative effect.

These are setup routines. The actual data base I/O is not invoked until a SEGHNLDR call is issued against the same segment ID. The settings remain in effect until a setup call with a different segment ID is issued or a RSETSEGH call is issued.

EXTENSIONS TO THE SEGHNDLR CALL

The full form of this call is as follows.

```
In COBOL:  CALL 'SEGHNDLR' USING ID, FUNC, KEY, OPER, PCBNO, AREA.
In PL/I:   CALL SEGHNDLR(ID,FUNC,KEY,OPER,PCBNO,AREA);
```

where:

PCBNO is a fullword (PL/I: BIN(31); COBOL: PICTURE S9(9) COMP.) giving the number of the user data base PCB in the PSB. The number can be from 1 to 120. If this parameter is not used, the default is the PCBNO operand value from the Rules Generator SEGMENT statement. Alternatively, the actual PCB address (not a PL/I pointer) may be passed in this parameter.

AREA is an I/O area defined in the SPR's working storage. If the segment requested is not a DBPATH or TSEGS segment, this parameter must be provided. Otherwise it is optional and the segment area in the SPA will be used if AREA is not present.

When any of the six parameters is coded, all parameters that precede it must also be coded.

The Get Next within Parent DL/I calls are supported with SEGHNDLR. However, they are worth using only in conjunction with the extended setup calls, since otherwise all SSAs above the lowest level are qualified. The following function codes are allowed:

GNP - Get Next within Parent, lowest level SSA unqualified.
GNPQ - Get Next with Parent, lowest level SSA qualified.
GHNP - Get Hold Next within Parent, lowest level SSA unqualified.
HNPQ - Get Hold Next within Parent, lowest level SSA qualified.

Two additional functions can be given in a SEGHNDLR call. The format of the call is altered to be the following:

```
In COBOL:  CALL 'SEGHNDLR' USING ID, FUNC, PCBNO[,AREA].
In PL/I:   CALL SEGHNDLR (ID,FUNC,PCB#[,AREA]);
```

The additional functions are:

```
'GU1 ' GET UNIQUE - position to first segment in the data base
'SGN ' GET NEXT   - sequential get next processing
```

No SSAs are passed to IMS/VS for the GU1 or SGN function if the ID is blank (' '). When the call completes successfully, the Input Transaction Rule is scanned to see if the retrieved segment is defined in it. If so, the ID is passed back to the user; otherwise, the ID field is blanked.

For the SGN function, if the ID is not blank and the segment is defined in the Input Transaction Rule, one unqualified SSA is passed to IMS/VS for the data base call. This SSA contains the DBD segment name (defined to IMSADF II by the Rules Generator parameter, NAME). This allows sequential processing of a single segment type. The SGN function is useful for a report application, where the SGN call could be followed by a MAPPER call to access pertinent data base fields.

A status code of GA or GK may be returned on a SGN call or on a call after a SETSSA. The return code to the user will be zero in these cases since a segment is returned to the caller.

The segment handler returns the following return codes in SPARTNCD:

Code	Meaning
0	DL/I call successfully completed.
4	Nonblank status code from IMS/VS. The status has been placed in SPADLIST. The appropriate error message number that is specified by the Segment Handler Rule is placed in the COMSG field of the segment handler communication area. A return code to the transaction driver of 28 will cause this message to be displayed or printed.

SET COMMAND CODES

SETCC is used to set one or two command codes per data base level. If a path call command code is encountered, the SPA segment work area cannot be used for data base I/O. A check is made at SEGHNDLR call time to make sure that the caller supplies an area for the data base I/O if any path call command codes were given in the SETCC call. The SETPATH call can be used in conjunction with the SETCC call in order to use the SPA segment work area for path calls.

Note: This allows three command codes per data base level.

The format of the SETCC call and its parameters is:

```
In COBOL:  CALL 'SETCC' USING TID, ARRAY1 [,ARRAY2].
In PL/I:   CALL SETCC (TID,ARRAY1[,ARRAY2]);
```

where:

TID is the target segment ID (i.e., the target of this call, not of the transaction). A Segment Handler Rule must exist for this segment.

ARRAY1 is a 15-byte array that contains the first command code for up to 15 data base levels. A blank or binary zero indicates that a command code is not to be used on the associated level.

ARRAY2 is the same as ARRAY1 except that this array is for the second command code. This parameter is optional.

SET PATH CALLS

SETPATH is used to set up a path call operation against a target segment that is defined in the Input Transaction Rule. If all of the requested segment IDs are in a defined path of loadable segments, the SPA segment area can be used for the data base I/O area. Otherwise, an area must be supplied in the subsequent SEGHNDLR call. When the SPA segment area is used, any loadable segments in the defined path that are not indicated in a SETPATH parameter will be retrieved on retrieval calls. This is done to keep the correct position of each segment in the segment area. This means that only the target segment ID must be given in order to retrieve a defined path.

The format of the call is:

```
  In COBOL:  CALL 'SETPATH' USING TID [, ID1, ...IDn].
  In PL/I:   CALL SETPATH (TID[,ID1,...IDn]);
```

where:

TID is the target segment ID.

ID1-IDn are IDs of other segments in the path that require some action.

A return code of 4 indicates that the SPA segment area cannot be used for the data base I/O.

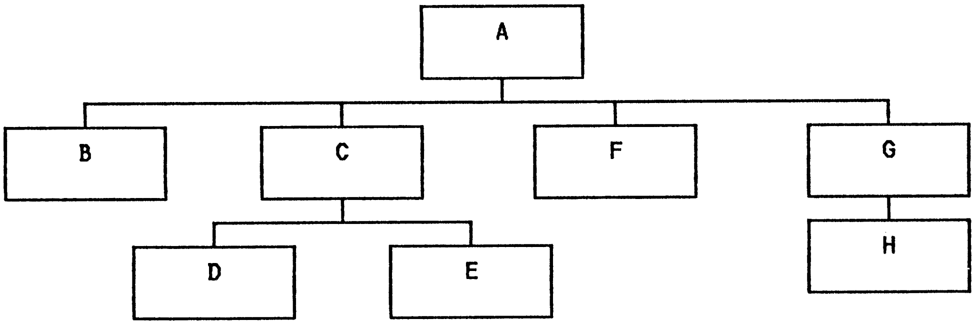


Figure 8-5. Data Base for Defined Paths Examples

The defined paths are controlled by the DBPATH and TSEGS parameters on the GENERATE statement of the Rules Generator. The following examples show what the defined paths will be for the segments shown in Figure 8-5.

Examples

1. DBPATH=(B,D,E),TSEG=(F,H)

   The paths, as defined to IMSADF II, will be:

   1-A,B 2-C,D 3-E 4-F 5-G,H

2. DBPATH=(E,B,D)

   The paths, as defined to IMSADF II, will be:

   1-A,C,E 2-B 3-D

3. DBPATH=(C,D,A,B,E)

   The paths, as defined to IMSADF II, will be:

   1-A,C,D 2-B 3-E

In example 1, if either segment A or C has no displayable data, it will become a key-only segment in the Input Transaction Rule (no segment space is reserved for it in the SPA segment work area). But, if a segment is listed in DBPATH or TSEGS, space for that segment will be reserved in the segment area. I/O in the SPA segment area is allowed only when all requested segments are contained within one path and they are not key-only segments.

In example 2, a SETPATH(D,C,A) would result in segment area I/O not being allowed (return code of 4). But, the path could be accessed by using an I/O area in the user's SPR.

Caution: If a delete is done immediately after a path retrieval that HOLDs segments, the following steps should be taken to ensure that the correct segment is deleted:

1. Issue a SETPATH call that specifies only the ID of the segment to be deleted.

2. Issue a SEGHNDLR call specifying the same ID and the DLET function.

SET SEGMENT SEARCH ARGUMENTS

Segment search arguments for DL/I calls can be set up for SEGHNDLR calls through the SETSSA routine. When the segment name in the last SSA matches a segment name in a Segment Layout Rule loaded for this transaction, the SPA segment area can be used for data base I/O as long as path call command codes are not present in the user's SSAs. In order for a Segment Layout Rule to be loaded, it must be defined in the Input Transaction Rule. When a match cannot be made on the last SSA segment name, the user is notified via a return code and the TID field is blanked out. Also, I/O cannot be done in the segment area.

The call format is:

```
  In COBOL:  CALL 'SETSSA' USING TID, SSA1[, SSA2, ..., SSAn].
  In PL/I:   CALL SETSSA(TID,SSA1[,SSA2,...,SSAn]);
```

where:

TID is the target segment ID. Will be set to blanks on return if not found in the Input Transaction Rule.

SSA1...SSAn are DL/I segment search arguments to be used in subsequent SEGHNDLR calls. These must conform to IMS/VS DL/I specifications. They are not validated by IMSADF II.

Each time this routine is called, all previous setups (of any type) are cleared and only the effects of the last call will apply to the next SEGHNDLR call. Use the alternative form of the SEGHNDLR call:

```
  In COBOL:  CALL 'SEGHNDLR' USING ID, FUNC, PCBNO[, AREA].
  In PL/I:   CALL SEGHNDLR(ID,FUNC,PCBNO[,AREA]);
```

The return codes for SETSSA are:

Code	Meaning
0	Successful completion.
4	TID blanked out, input value not found in Input Transaction Rule.
8	TID reset to match ID associated with Segment Layout Rule that matched segment name in last user SSA. This occurs only if the TID and SSA segment names do not correspond.

SET UNQUALIFICATION

SETUNQ is used to unqualify segment search arguments at desired levels. The segment name and any command codes that have been set up will not be affected by this routine. The unqualification takes place where the begin qualification character, '(', would appear in the SSA, i.e., after any command codes that SETCC has set up.

The call format is:

```
  In COBOL:  CALL 'SETUNQ' USING TID, ARRAY.
  In PL/I:   CALL SETUNQ(TID,ARRAY);
```

where:

TID is the target segment ID. A Segment Handler Rule must exist for this ID.

ARRAY is an array of 15 characters that corresponds to levels in a data base.

The character 'U' will cause the associated level to be unqualified on subsequent SEGHNDLR calls. If any other character is specified, the level will be qualified or unqualified in the usual manner (controlled by the function given in SEGHNDLR call).

RESET CALL

RSETSEGH clears any previous data base setup calls. This is done automatically in the following cases:

- A new TID in either a set call or a SEGHNDLR call.
- A SETPATH call after a SETCC call that contained path call command codes.
- A SETSSA call.
- SETCC, SETPATH, or SETUNQ call after a SETSSA call.
- On a subsequent SETCC, SETPATH, or SETUNQ call.

The format is:

```
  In COBOL:  CALL 'RSETSEGH'.
  In PL/I:   CALL RSETSEGH;
```

No parameters are required.

PROGRAM LINKAGE

A special processing routine is called by the transaction driver and must therefore be named according to IMSADF II standards. The name has the form:

ssssUtx

where:

ssss is the application system ID
U is a literal
tx is the transaction ID

When the program has been compiled, it should be link-edited with an interface module, known as a mini-driver, which dynamically links to the transaction driver at execution time. This service will be performed by the Rules Generator in a special run. JCL and control statement examples are given below.

For COBOL:

```
//COMPILE EXEC   COBUC,PARM.COB='BUF=40K,DECK,NOLOAD,NORESIDENT,ENDJOB,NODYN'
//COB.SYSPUNCH DD DSN=IMSADF.PROGRAM.OBJ(SAMPUTT),DISP=OLD
//COB.SYSLIB   DD DSN=IMSADF.MACLIB.ASM,DISP=SHR
         IDENTIFICATION DIVISION.
          PROGRAM-ID
             SAMPUTT.
         ..etc.
//LKED    EXEC ????G
//G1.SYSLIB  DD
//            DD
//            DD DSN=SYS1.COBLIB,DISP=SHR
       SYSTEM  SYSID=SAMP
       GENERATE OPT=SPLE,PGMID=TT,MAPTABLE=(MM,MO),
                SHTABLE=PD
```

For PL/I:

```
//COMPILE EXEC PLIXC
//PLI.SYSPUNCH DD DSN=IMSADF.PROGRAM.OBJ(SAMPUTT),DISP=OLD
//PLI.SYSLIB   DD DSN=IMSADF.MACLIB.ASM,DISP=SHR
 *PROCESS X, NEST, DECK, OFFSET, OPT(2), SIZE(MAX), MACRO
  SAMPUTT:  PROCEDURE (SPA);
  DCL PLIXOPT CHAR(50) VAR STATIC EXTERNAL
      INIT('NOCOUNT,NOFLOW,NOREPORT,NOSTAE,NOSPIE);
  DCL (AUDITOR,MAPPER,SEGUPDTE,SEGHNDLR) ENTRY OPTIONS(ASSEMBLER);
      /* ALL SUBROUTINES USED MUST BE DECLARED THUS */
  ..etc.
//LKED        EXEC ????G
//G1.SYSLIB  DD
//            DD
//            DD DSN=SYS1.PLIBASE,DISP=SHR
       SYSTEM SYSID=SAMP
       GENERATE OPT=SPLE,PGMID=TT,MAPTABLE=(MM,MO)
                SHTABLE=PD,LANG=PL/I
//
```

Notes:

1. **????** is the installed ADFID (the default is MFC1).

2. These JCL examples assume that the RGLIB=O option was used on the DEFADF macro during installation (see the IMS Application Development Facility II Version 2 Release 2 Installation Guide). If your installation uses RGLIB=L, you must link-edit the program into IMSADF.RULLIB before the Rules Generator link-edits it again.

In the above samples, the transaction ID is TT. The GENERATE statement operands or values are as follows:

OPT=SPLE special processing link-edit (conversational transaction).

PGMID the transaction ID.

MAPTABLE mapping segment IDs quoted in MAPPER calls.

SHTABLE segment IDs quoted in SEGHNDLR calls.

If MAPTABLE or SHTABLE is not coded, the system will load the corresponding rules at the time they are used. This is preferable when testing mapping segment rules to avoid the need to relink-edit when changes are necessary. In production use MAPTABLE and SHTABLE, unless the mapping rules and Segment Handler Rules are in the IMSADF II preload table.

LINKAGE CONVENTIONS

When the SPR receives control, it is passed a number of parameters but will normally require only the first, which is the SPA, used as a general communication and work area by IMSADF II. Here is the complete list of parameters passed:

- SPA
- segment handler communication area (COMOPT)
- Audit data base PCB
- Message data base PCB
- application system data base PCBs

There will be as many application system data base PCBs as are present in the PSB. Following them, for conversational and nonconversational programs, will be the following additional PCBs:

- I/O terminal PCB
- alternate terminal PCB
- express terminal PCB

The programming conventions to receive the SPA are now given:

In COBOL:

```
LINKAGE SECTION.
    COPY SPACOBOL.
PROCEDURE DIVISION USING SPADSECT.
```

In PL/I:

```
SAMPUTT: PROCEDURE (SPA);
DCL SPA BIN(31);
%INCLUDE SPAPLI;
SPAPTR=ADDR(SPA);
```

SPA FIELDS

A number of fields are provided to enable the SPR to communicate with the transaction driver and with subroutines.

Field	Size	Description
SPAFIRST	Half Word Binary	Set to zero by the transaction driver on the initial call to the SPR for a transaction. Since the SPR can iterate with the driver any number of times by appropriate setting of the return code, this field can be used to keep track of which iteration is being processed. The batch driver also sets SPAFIRST to (-1) on the End-of-File call.

Field	Size	Description
SPARTNCD	Full Word Binary	Contains the return code from subroutine calls; is not used to pass a return code to the transaction driver.
SPADLIST	Character 2	Contains the DL/I status code returned from a SEGHNDLR call.
SPAERMSG	Character 50	Can be used to display a program-generated message on the message line of the Data Display screen in conversational or nonconversational processing. The program can place a message in this field and return to the transaction driver with a return code of 3. This causes the Data Display screen to be displayed with that message. When the batch transaction driver is in control, the message will be printed on the system printer. When the nonconversational driver is operating as a nonresponse transaction, the message will be printed on a 3284/3286 printer.
SPAKEYID	Character 255	Contains the fully concatenated keys (as displayed) of all target segment paths (DBPATH) built during key selection or entered with the transaction in nonconversational or batch processing. The program may use portions of this key to access other data base segments in SEGHNDLR calls.
SPASECTX	Halfword Binary	Requests that a secondary transaction be generated by the conversational or nonconversational driver. Figure 8-6 shows how the setting of SPASECTX ties in with the coding of the STX operand of the GENERATE statement. (In this figure, NO means a transaction was not sent; YES means a transaction was sent.) In standard processing, the transaction driver sends secondary transactions for STX=OK at normal termination and the STX=ER ones on abnormal termination. In special processing, the program determines when the error transactions will be sent and when the normal ones will go.
SPACGTRX	Character 3	Contains the next standard or special processing transaction ID that will be processed by the conversational transaction driver. The special processing routine sets this to the ID that logically steps the end user to the next function to be performed. The format of SPACGTRX is MXX where M is mode and XX is the ID. Entering the mode and ID in this field and returning to the transaction driver (with a return code of 5) will cause IMSADF II to proceed as if the terminal user had modified the TRX field on the Data Display screen. At the same time, the program can set SPAKEYID with the same effect as the end user modifying the KEY field on the screen.
SPAMANNO	Character 6	User ID signed on.
SPALTERM	Character 8	User's logical terminal name.
SPAPROJ	Character 1	Project signed on.

Field	Size	Description
SPAGROUP	Character 1	Group signed on.
SPATRXCD	Character 1	Current transaction mode (1 to 6).
SPAUTILY	Character	A programmer-maintained area in the SPA, the length of which is contained in the field SPACOMLN, and is defined by the COMMLEN keyword on the GENERATE statement. This area is preserved across conversational transaction switches. If a subsequent transaction requires a larger area, the current area will be extended. If a subsequent transaction requires a smaller area, the length remains unchanged.
SPAFLDSG	Character	Same offset in the SPA as SPAUTILY. Can be used to access the programmer-maintained area in the SPA.
SPACOMLN	Halfword Binary	The length of the area addressed by SPAUTILY. If this field contains binary zeros, no programmer-maintained area exists in the SPA.

SPASECTX	STX=OK	STX=ER	STX=(OK,ER)	Signalled by Auditor (STX not OK or ER)
= 0	NO	NO	NO	NO
= 1	YES	NO	YES	YES
= 2	NO	YES	YES	YES
= 3	YES	YES	YES	YES

Figure 8-6. Control of Secondary Transaction Sending Via SPASECTX

RETURN CODES

The special processing routine sets a return code to the transaction driver to tell it what to do next. The return code must be set through the usual operating system method (COBOL: RETURN-CODE, PL/I: PLIRETC, Assembler: Register 15), not in SPARTNCD. The following return codes apply to the conversational transaction driver. Batch and nonconversational SPR return codes are listed in Chapter 10, "Batch Processing" and Chapter 11, "Nonconversational Processing."

Code	Transaction Driver Action
0	Return to Primary Option Menu.
1	Perform automatic message sending and send secondary transactions (according to the setting of SPASECTX), then call the SPR again. Using this return code, the SPR can call the Auditor and trigger message sending several times during one execution of the transaction. See "Multiple Iterations of Message Sending" on page 8-27 for an example.
2	Display the screen. If the user enters amendments, recall the SPR; this process can be repeated as often as necessary.
3	Display the screen with the message that the SPR has placed in SPAERMSG. As for return code 2, the user can enter amendments, causing the SPR to be called again.
4	Display the screen with the message SPECIAL PROCESSING SUCCESSFULLY EXECUTED and continue as for return code 2 and 3.

If return codes 2, 3 or 4, which are normal returns, are used, the SPR must be capable of being executed more than once if the user enters amendments more than once. The SPR can keep a count of the number of times it has been called if SPAFIRST is used. Every time the user changes the concatenated key or the transaction mode or ID on the screen, the transaction driver will reset SPAFIRST to zero. This way, the SPR can tell whether it is in the first or a subsequent iteration.

5 Switch to the new transaction mode and ID placed in SPACGTRX using the concatenated key in SPAKEYID. The present transaction is terminated without an acknowledgement to the end user. The next thing he sees is the Key Selection or Data Display screen for the new transaction. The security profile is checked and the user will be shown the Secondary Option Menu if he is not authorized to use the new transaction, which may be standard or special processing with the same or a different cluster code.

8 Display the screen with the message ENTER "E" TO DISPLAY ERROR MESSAGES and highlight fields in error after auditing. If the user enters E, display the error messages without calling the SPR. When the user enters amendments, call the SPR, which should recall the AUDITOR.

12 The SPR has sent messages to the screen using the DISPLAYL call. The operation is similar to the display of error messages after the user has entered E. The first page of messages, with heading, will be displayed; the user can obtain subsequent pages by pressing PA1. When he presses the ENTER key, the Data Display screen will appear.

24 Issue a DL/I ROLL call to back out any data base updates performed by the SPR during this execution. The transaction is terminated, the user receives a message and he must sign on again.

28 The SPR has received a bad return code from a SEGHNDLR or SETUPDTE call. Display the error message set up by the segment handler. When the user sees the message, all he can do is press ENTER to return to the Primary Option Menu.

32 The SPR has sent messages to the screen using the DISPLAYL call but, unlike return code 12, is terminating the transaction. The user can page using PA1, but when he presses ENTER, he returns to the Primary Option Menu. The processing with return code 28 and 32 is similar to that used to display error messages during preaudit, where the user is prevented from viewing the data.

SPECIAL PROCESSING EXAMPLES

In these examples, a basic program that reproduces the functions of standard processing (apart from transaction modes 1 to 3) is presented after a specific example is given.

BASIC SPECIAL PROCESSING PROGRAM

COBOL

```
PROCEDURE DIVISION USING SPADSECT.
AUDIT.
    CALL 'AUDITOR'.
    IF SPARTNCD > 4 THEN GO TO RC-8-GOBACK. NOTE AUDIT ERRORS.
SEGUPDTE.
    CALL 'SEGUPDTE'.
    IF SPARTNCD = 0 THEN PERFORM NORMAL-MSG.
    IF SPARTNCD = 4 THEN GO TO RC-28-GOBACK. NOTE DB ERRORS.
    IF SPARTNCD = 12 THEN GO TO DATACOMP.
    IF SPARTNCD = 16 THEN
      MOVE '*** NO MODIFICATION MADE ***' TO SPAERMSG.
    MOVE 3 TO RETURN-CODE.
    GOBACK.
NORMAL-MSG
    IF SPATRXCD = '4'
      MOVE '*** SEGMENT ADDED SUCCESSFULLY ***'
        TO SPAERMSG.
    IF SPATRXCD = '5'
      MOVE '*** SEGMENT MODIFIED SUCCESSFULLY ***'
        TO SPAERMSG.
DATACOMP.
    MOVE 'SOMEONE HAS CHANGED THE DATA - PLEASE RESTART'
      TO SPAERMSG.
    MOVE 3 TO RETURN-CODE.
    GOBACK. NOTE USER MUST ALTER TRX OR KEY OR ENTER OPTION C.
RC-8-GOBACK.
    MOVE 8 TO RETURN-CODE.
    GOBACK. NOTE ALLOW USER TO CORRECT ERROR.
RC-28-GOBACK.
    MOVE 28 TO RETURN-CODE.
    GOBACK.  NOTE DISPLAY ERROR MSG AND GO TO P O M.
```

PL/I

```
SPAPTR=ADDR(SPA);
AUDIT:  CALL AUDITOR;
 IF SPARTNCD > 4 THEN GOTO RC_8_RETURN;/*AUDIT ERRORS*/
SEGUPDTE: CALL SEGUPDTE;
 IF SPARTNCD=0 THEN DO;
   IF SPATRXCD='4' THEN
     SPAERMSG='*** SEGMENT ADDED SUCCESSFULLY ***';
   IF SPATRXCD='5' THEN
     SPAERMSG='*** SEGMENT MODIFIED SUCCESSFULLY ***';
   END;
 IF SPARTNCD=4 THEN GOTO RC_28_RETURN;/*DB ERRORS*/
 IF SPARTNCD=12 THEN GOTO DATACOMP;
 IF SPARTNCD=16 THEN
   SPAERMSG='***NO MODIFICATIONS MADE***';
 CALL PLIRETC(3); RETURN;
DATACOMP: SPAERMSG='SOMEONE HAS CHANGED THE DATA - PLEASE RESTART';
 CALL PLIRETC(3); RETURN;
 /* USER MUST ALTER TRX OR KEY OR CHANGE OPTION */
RC_8_RETURN:  CALL PLIRETC(8); RETURN;
 /* ALLOW USER TO CORRECT ERROR */
RC_28_RETURN: CALL PLIRETC(28); RETURN;
 /* DISPLAY ERROR MESSAGE AND GO TO P.O.M. */
 END;
```

The above program is not called prior to segment display and BYPASS=NO must be coded on the GENERATE statement for the transaction. The program is given here as a basis from which to write special processing routines.

MULTIPLE ITERATIONS OF MESSAGE SENDING

Sometimes it may be necessary to invoke automatic message sending or send a secondary transaction several times during the course of executing an SPR. For example, a program may retrieve several segment occurrences (twins), call the Auditor to trigger message or transaction generation and then replace the segment. Messages and transactions are sent only when the SPR returns to the transaction driver. It is therefore necessary to make repeated uses of return code 1 to ask the transaction driver to perform the sending and immediately call the SPR again. The SPAFIRST field must be used to keep track.

In COBOL:

```
       MOVE 1 TO SPASECTX. NOTE SECONDARY TRANSACTION CONDITIONS.
   LOOP.
       CALL 'SEGHNDLR' USING SG, GHN.
       IF SPADLIST = GE GO TO LOOP-END. NOTE SEG NOT FOUND.
       IF SPARTNCD > 0 GO TO RC-28-GOBACK.
       PERFORM MANIP. NOTE DATA MANIPULATION (NOT SHOWN).
       CALL 'AUDITOR'.
       IF SPARTNCD > 4 GO TO RC-8-GOBACK. NOTE ERRORS.
       IF SPARTNCD = 4 GO TO RC-1-GOBACK. NOTE MSGS.
   RESUME.
       MOVE 1 TO SPAFIRST.
       CALL 'SEGHNDLR' USING SG, REPL.
       IF SPARTNCD > 0 GO TO RC-28-GOBACK.
       GO TO LOOP.
   RC-1-GOBACK.
       MOVE 100 TO SPAFIRST.
       MOVE 1 TO RETURN-CODE. GOBACK.
   LOOP-END.
   etc.
```

The very first statement in the PROCEDURE DIVISION must be:

```
       IF SPAFIRST=100 GO TO RESUME.
```

The WORKING-STORAGE definitions required are:

```
          77 SG   PICTURE XX   VALUE 'SG'.
          77 GHN  PICTURE X(4) VALUE 'GHN '.
          77 GE   PICTURE XX   VALUE 'GE'.
          77 REPL PICTURE X(4) VALUE 'REPL'.
```

In PL/I:

```
     SPASECTX=1; /*SECONDARY TRANSACTION CONDITIONS*/
    LOOP:  CALL SEGHNDLR ('SG','GHN ');
     IF SPADLIST='GE' THEN GOTO LOOP_END;/*SEG NOT FOUND*/
     IF SPARTNCD > 0 GOTO RC_28_RETURN;
     CALL MANIP;   /* DATA MANIPULATION SUBROUTINE*/
     CALL AUDITOR;
     IF SPARTNCD > 4 THEN GOTO RC_8_RETURN; /*ERRORS*/
     IF SPARTNCD = 4 THEN GOTO RC_1_RETURN; /*MSGS*/
    RESUME:  SPAFIRST = 1;
     CALL SEGHNDLR('SG','REPL');
     IF SPARTNCD > 0 THEN GOTO RC_28_RETURN;
     GOTO LOOP;
    RC_1_RETURN: SPAFIRST=100;
     CALL PLIRETC(1); RETURN;
    LOOP_END:
    ..etc.
```

The very first statement in the program must be:

```
     IF SPAFIRST=100 THEN GOTO RESUME;
```

IMS/VS CONSIDERATIONS

Each special processing transaction ID must have an associated IMS/VS transaction code with name

ssssVtx

where:

ssss is the application system ID
V is a literal
tx is the transaction ID

The Rules Generator GENERATE OPT=SPLE statement will produce an executable load module with the same name (ssssVtx). A PSB with the same name must also be provided. Apart from the difference in name, the coding of the PSB and IMS/VS system definition macros is the same as for standard processing. Brief details appear in Chapter 2, "Static Rules and the Rules Generator." For more information consult the *IMS Application Development Facility II Version 2 Release 2 Application Development Reference*.

There is an alternative convention. Refer to the discussion of the GENERATE statement DYNAMIC=YES option in the *IMS Application Development Facility II Version 2 Release 2 Application Development Reference*.

CHAPTER 9. EXITS

IMSADF II provides exits that allow you to enhance the capabilities of functions supplied with the product or to tailor functions to your particular requirements.

AUDITOR EXIT ROUTINES

You can extend the capabilities of the Auditor by supplying routines that perform additional operations. Such operations may include performing calculations involving many fields and handling DL/I calls not supported by the Auditor DL/I call facility, such as Get Last and Get Previous (using SETCC calls to set command codes).

To invoke an exit routine with the high level audit language, use the AEXIT keyword, followed by one of the operation codes from 70 to 99 and from W0 to Z9 that are reserved for exit routines. Since an exit routine returns a true or false indicator, it is invoked within an IF statement. For example:

```
  IF AEXIT 75 RETURN = FALSE
    ABCD = 1
    ENDIF
```

This will cause the Auditor to call the routine to perform the desired function. The exit routine must return to the Auditor with a true, false, or error indicator. The Auditor will continue with the next statement, which can be NOP (no operation) or any other statement in the language, according to the logic of the IF statement in the case of a true or false return. If the exit sets an error indicator, the Auditor will terminate processing for that field and go on to the audit rules for the next field (if any).

The exit routine can be written in COBOL, Assembler or PL/I and is entitled to use all the subroutine calls available to a special processing routine except for the AUDITOR and the DISPLAYL calls. It has access to the SPA and should use those communication fields (SPARTNCD, SPADLIST, etc.) in the same way. Linkage, however, is different since the program is called by the Auditor, not the transaction driver. The exit routine receives a different set of parameters and does not set a return code. Hence, it cannot alter the transaction flow to the same extent as an SPR. In particular, it cannot request a return to the Primary Option Menu; it cannot set any return codes (other than the true/false indicator); and it cannot request a switch to a new transaction ID.

The exit routine can, however, access and update data fields using the MAPPER and therefore is not restricted to accessing an audited or a related field.

In addition, it can mark fields in error using the SETERROR call. This has the same effect as the ERRORMSG statement in the high level audit language but is not restricted to the audited field. Refer to the IMS Application Development Facility II Version 2 Release 2 Application Development Reference for information on the SETERROR call.

PARAMETERS

The following parameters are passed to the exit routine:

- The audited field
- The audited field's descriptor in the Segment Layout Rule
- The audit operation code - a 2-byte code (70 to 99 or W0 to Z9)
- The Audit Data Base PCB used to retrieve data descriptors
- COMOPT
- The true/false indicator
- The function indicator telling which of the three Audit data base legs is being interpreted
- The scratch pad area
- A list of application data base PCBs in Assembler language format
- The concatenated key of the audit operation descriptor segment
- The related field, if specified in the AEXIT statement (or in the operation descriptor); otherwise, the audited field is passed again
- The related field descriptor in the Segment Layout Rule or a repeat of that for the audited field if there is no related field
- The data descriptor area
- 3 dummy parameters
- Addr(IOPCB)
- Addr(ALTIOPCB)
- Addr(EXPIOPCB)
- Addr(count of user data base PCB's)
- Addr(user PCB 1)

  .
  .
  .
- Addr(user PCB 120)

A related field, if required, is specified in the AEXIT statement of the high level audit language. For example:

```
  IF AEXIT 75 SAPDMKDP RETURN = FALSE
    ABCD = 3
    ENDIF
```

Sample coding to receive these parameters appears in "Sample Audit Exit Routines" on page 9-3.

In PL/I a standard prologue for PL/I audit exit routines can be defined as shown in Figure 9-1 (the sample in the next section includes the prologue using the statement: **%INCLUDE AUDEXSET;**).

```
/*INCLUDED TEXT - SET UP FOR USER WRITTEN AUDIT EXIT IN PL/I */
/*MUST BE PRECEDED BY INCLUDE SPAPLI */
DCL (A1,A2,A3,A4,A5,A6,A7,A8,A9,A10,A11,A12,A13) BIN FIXED(31);
DCL AUDITED_FIELD CHAR(17) BASED(P1), /*RE-DEFINE AS NECESSARY */
    FIELD_DESC CHAR(1) BASED(P2),
    OPCODE CHAR(2) BASED(P3),
    AUDIT_PCB CHAR(1) BASED(P4),
    COMOPT CHAR(1) BASED(P5),  /* could be CHAR(130) */
    TRUE_FALSE_ERROR BIT(8) BASED(P6),
    FUNCTION_INDIC CHAR(1) BASED(P7),
    PCBLIST CHAR(1) BASED(P9),
    COKEY CHAR(1) BASED(P10),
    RELATED_FIELD CHAR(17) BASED(P11), /* RE-DEFINE AS NECESSARY */
    RELATED_FIELD_DESC CHAR(1) BASED(P12),
    01 DATADESC BASED(P13),
      02 DDLEN BIN(31),
      02 AREA,
        03 DDSEQNO CHAR(4),
        03 DDDATA CHAR(24); /* DATA DESCRIPTOR DATA */
DCL (SEGHNDLR,GETKEY,MAPPER) ENTRY OPTIONS(ASSEMBLER);
DCL PLIXOPT CHAR(50) VAR STATIC EXTERNAL
    INIT('NOCOUNT,NOFLOW,NOREPORT,NOSTAE,NOSPIE');
DCL (FROM_WORKAREA INIT(1), TO_WORKAREA INIT(0)) BIN FIXED(31);
DCL (TRUE INIT('1'B), FALSE INIT('0'B), ERROR INIT('11'B)) BIT(8);
    P1=ADDR(A1);
    P2=ADDR(A2);
    P3=ADDR(A3);
    P4=ADDR(A4);
    P5=ADDR(A5);
    P6=ADDR(A6);
    P7=ADDR(A7);
    SPAPTR=ADDR(A8);
    P9=ADDR(A9);
    P10=ADDR(A10);
    P11=ADDR(A11);
    P12=ADDR(A12);
    P13=ADDR(A13);
    TRUE_FALSE_ERROR = TRUE;
/* APPLICATION CODING BEGINS HERE */
```

Figure 9-1. Standard Prologue for PL/I Audit Exit Routines

SAMPLE AUDIT EXIT ROUTINES

This sample application continues one begun in "Transaction Switching" on page 6-9. If the receiving transaction is in standard processing, it will need an audit exit to accept the message information passed to it and move it into appropriate fields in the SPA by means of a mapping segment.

JCL is included in the samples below. The subroutines are written to the static rules load library, which is the most suitable place from which to combine them with the Auditor.

COBOL Routine

```
//EXEC COBUCL,PARM.COB='BUF=40K,DECK,NOLOAD,
//      APOST,NOSEQ,LIB,NORES,NODYN,NOENDJOB',
//      PARM.LKED='REUS,NCAL,LET,LIST,XREF'
//COB.SYSLIB DD DSN=IMSADF.ADFMAC,DISP=SHR
//COB.SYSIN DD *
       IDENTIFICATION DIVISION.
        PROGRAM-ID.
           AUDEX70.
        DATE-COMPILED. JUNE 19,1982.
        REMARKS.
           AUDIT EXIT SUBROUTINE TO RECEIVE INFORMATION FROM A
           TRANSACTION SWITCH.
        ENVIRONMENT DIVISION
         CONFIGURATION SECTION.
          SOURCE-COMPUTER. IBM-370.
          OBJECT-COMPUTER. IBM-370.
        DATA DIVISION.
        WORKING-STORAGE SECTION.
        77 MAP-INFO-IN    PICTURE XX VALUE 'OM'. NOTE MAPPING SEGMENT.
        77 TO-WORKAREA    PICTURE S9(9) COMP VALUE 0.
        77 FROM-WORKAREA  PICTURE S9(9) COMP VALUE 1.
        77 FALSE          PICTURE X VALUE LOW-VALUES.
        01 TRUEINIT       PICTURE S9(9) COMP VALUE 128.
        01 TRUEDEF        REDEFINES TRUEINIT.
           02 FILLER      PICTURE XXX.
           02 TRUE        PICTURE X.
        01 ERINIT         PICTURE S9(9) COMP VALUE 192.
        01 ERDEF          REDEFINES ERINIT.
           02 FILLER      PICTURE XXX.
           02 ER          PICTURE X.
        LINKAGE SECTION.
        77 AUDITED-FIELD  PICTURE X(17). NOTE REDEFINED AS NECESSARY.
        77 FIELD-DESC     PICTURE X.
        77 AUDIT-DESC     PICTURE XX.
        77 AUDIT-PCB      PICTURE X.
        77 COMOPT         PICTURE X.
        77 TRUE-FALSE-ER  PICTURE X.
        77 FUNCTION-INDIC PICTURE X.
        77 PCBLIST        PICTURE X.
        77 COKEY          PICTURE X.
        77 RELATED-FIELD  PICTURE X(17). NOTE REDEFINE AS NECESSARY.
        77 RELATED-FIELD-DESC PICTURE X.
        01 SPADSECT COPY SPACOBOL.
        01 DATADESC.
          02 DDLEN        PICTURE S9(9) COMP.
          02 DDAREA       OCCURS XX. NOTE XX SHOULD BE AT LEAST AS
                          LARGE AS THE MAXIMUM # OF DATA DESC.
            03 DDSEQNO    PICTURE XXXX.
            03 DDDATA     PICTURE X(24). NOTE DATA DESCRIPTOR DATA.
        PROCEDURE DIVISION USING AUDITED-FIELD, FIELD-DESC, AUDIT-DESC,
            AUDIT-PCB, COMOPT, TRUE-FALSE-ER, FUNCTION-INDIC, SPADSECT,
            PCBLIST, COKEY, RELATED-FIELD, RELATED-FIELD-DESC,
            DATADESC.
            MOVE TRUE TO TRUE-FALSE-ER.
            CALL 'MAPPER' USING MAP-INFO-IN, SPAFLDSG, FROM-WORKAREA.
            IF SPARTNCD   0 MOVE FALSE TO TRUE-FALSE-ER.
//LKED.SYSLMOD DD DSN=IMSADF.RULLIB(AUDEX70),DISP=OLD
```

PL/I Routine

```
// EXEC PLIXCL,PARM.LKED='REUS,NCAL,LET,LIST,XREF'
//PLI.SYSLIB DD DSN=IMSADF.ADFMAC,DISP=SHR
//PLI.SYSIN DD *
* PROCESS X,NEST,LOAD,OFFSET,OPT(2),SIZE(MAX),MACRO;
 AUDEX70: PROC(A1,A2,A3,A4,A5,A6,A7,A8,A9,A10,A11,A12,A13);
  /* SAMPLE AUDIT EXIT */
 %INCLUDE SPAPLI;
 %INCLUDE AUDEXSET;
   CALL MAPPER ('OM',SPAFLDSG,FROM_WORKAREA);
   IF SPARTNCD > 0 THEN TRUE_FALSE_ERROR = FALSE;
   END;
//LKED.SYSLMOD DD DSN=IMSADF.RULLIB(AUDEX70),DISP=OLD
```

DATA DESCRIPTORS

When an audit exit is called, it can also receive data descriptor segments, which can be used to pass application specific parameters to it. The call to the exit is augmented with the PASS keyword in the high level audit language.

For example:

```
  IF AEXIT 70 PASS 'OM' RETURN = TRUE
    ABCD = 2
    ENDIF
```

This sample exit routine could be made to receive the ID of the mapping segment in this way instead of making it a literal in the program.

To retrieve data descriptor details, the exit routine must check to see if they have been passed in the 13th parameter (as they will be if static audit rules are in use) or if a SEGHNDLR call must be made. The count field at the beginning of the 13th parameter will be set to minus one if the data is to be retrieved from the data base.

In a data base retrieval, data may be retrieved from any of the three legs; therefore, the appropriate data descriptor segment ID (DA, DF or DM) must also be retrieved. Here is some sample coding:

In COBOL:

```
      IF FUNCTION-INDIC=1 MOVE 'DA' TO DD.
      IF FUNCTION-INDIC=2 MOVE 'DF' TO DD.
      IF FUNCTION-INDIC=3 MOVE 'DM' TO DD.
      IF DDLEN < 0
        CALL 'SEGHNDLR' USING DD, GUU, COKEY, FE, AUDIT-PCB, AREA.
  In WORKING-STORAGE SECTION.
      77 DD    PICTURE XX
      77 GUU   PICTURE X(4) VALUE 'GUU '.
      77 FE    PICTURE XX   VALUE 'FE'.
```

In PL/I:

```
      DCL DD CHAR(2);
      IF FUNCTION_INDIC=1 THEN DD='DA';
      IF FUNCTION_INDIC=2 THEN DD='DF';
      IF FUNCTION_INDIC=3 THEN DD='DM';
      IF DDLEN < 0 THEN
        CALL SEGHNDLR(DD,'GN  ',COKEY,'FE',AUDIT_PCB,AREA);
```

The DL/I status code (SPADLIST) should be checked for 'GE' (segment not found) when retrieving a variable number of data descriptors.

The other identifiers quoted in the above samples are parameters passed to the exit routine by the Auditor. The data descriptor value will be in the field DDDATA.

DESIGN AND LINK-EDIT OF AN AUDIT EXIT ROUTINE

Normally, a separate routine is written to handle each of the required operation codes (in range 70-99 and W0-Z9). The Auditor, however, invokes a single exit for all these operation codes. Therefore, you might want to write a general exit routine that examines the operation code passed to it and calls the appropriate subroutine. Such a routine is given below.

The sample routine shown allows for 10 subroutines, one for each of the codes 70 to 79. It is most common to provide a single audit exit routine for the entire installation, but you may have a different one in each application system, or even for each cluster code. The sample routine is accompanied by JCL that shows how to link-edit the exit routine with the Auditor. The routine is designed to allow new subroutines to be added by means of a link-edit up to the chosen limit of 10. Simply rerun the following job stream to include new modules to process different audit operation codes.

COBOL Routine

```
// EXEC COBUC,PARM.COB='BUF=40K,DECK,NOLOAD,NODYN,
//      APOST,NOSEQ,LIB,NORES,NOENDJOB'
//COB.SYSLIB DD DSN=IMSADF.ADFMAC,DISP=SHR
//COB.SYSIN DD *
       IDENTIFICATION DIVISION
        PROGRAM-ID.
           AUDEXIT.
        DATE-COMPILED.MAY 1,1979.
        REMARKS.
           AUDIT EXIT.
       ENVIRONMENT DIVISION.
        CONFIGURATION SECTION.
         SOURCE-COMPUTER.IBM-370.
         OBJECT-COMPUTER.IBM-370.
       DATA DIVISION.
       LINKAGE SECTION.
       77 A1                 PICTURE X.
       77 A2                 PICTURE X.
       77 OPCODE             PICTURE XX.
       77 A4                 PICTURE X.
       77 A5                 PICTURE X.
       77 A6                 PICTURE X.
       77 A7                 PICTURE X.
       77 A8                 PICTURE X.
       77 A9                 PICTURE X.
       77 A10                PICTURE X.
       77 A11                PICTURE X.
       77 A12                PICTURE X.
       77 A13                PICTURE X.
       PROCEDURE DIVISION USING A1, A2, OPCODE, A4, A5, A6, A7,
           A8, A9, A10, A11, A12, A13.
           IF OPCODE = '70' CALL 'AUDEX70' USING
             A1, A2, OPCODE, A4, A5, A6, A7, A8, A9, A10, A11, A12, A13.
           IF OPCODE = '71' CALL 'AUDEX71' USING
             A1, A2, OPCODE, A4, A5, A6, A7, A8, A9, A10, A11, A12, A13.
           IF OPCODE = '72' CALL 'AUDEX72' USING
             A1, A2, OPCODE, A4, A5, A6, A7, A8, A9, A10, A11, A12, A13.
           IF OPCODE = '73' CALL 'AUDEX73' USING
             A1, A2, OPCODE, A4, A5, A6, A7, A8, A9, A10, A11, A12, A13.
           IF OPCODE = '74' CALL 'AUDEX74' USING
             A1, A2, OPCODE, A4, A5, A6, A7, A8, A9, A10, A11, A12, A13.
           IF OPCODE = '75' CALL 'AUDEX75' USING
             A1, A2, OPCODE, A4, A5, A6, A7, A8, A9, A10, A11, A12, A13.
           IF OPCODE = '76' CALL 'AUDEX76' USING
             A1, A2, OPCODE, A4, A5, A6, A7, A8, A9, A10, A11, A12, A13.
           IF OPCODE = '77' CALL 'AUDEX77' USING
             A1, A2, OPCODE, A4, A5, A6, A7, A8, A9, A10, A11, A12, A13.
           IF OPCODE = '78' CALL 'AUDEX78' USING
             A1, A2, OPCODE, A4, A5, A6, A7, A8, A9, A10, A11, A12, A13.
           IF OPCODE = '79' CALL 'AUDEX79' USING
             A1, A2, OPCODE, A4, A5, A6, A7, A8, A9, A10, A11, A12, A13.
```

```
          IF OPCODE = '80' CALL 'AUDEX79' USING
            A1, A2, OPCODE, A4, A5, A6, A7, A8, A9, A10, A11, A12, A13.
// EXEC ????G
//G1.SYSLIB DD
//     DD
//     DD DSN=SYS1.COBLIB,DISP=SHR
  SYSTEM SYSID=SAMP,LOPTPARM='XREF,REUS,LET'
  GENERATE OPT=STLE,PGMID=OR,AEXIT=AUDEXIT
/*
```

Note: ???? is the installed ADFID (the default is MFC1).

PL/I Routine

```
// EXEC PLIXC
//PLI.SYSLIB DD DSN=IMSADF.ADFMAC,DISP=SHR
//PLI.SYSIN DD *
* PROCESS X,NEST,DECK,OFFSET,OPT(2), SIZE(MAX),MACRO;
 AUDEXIT: PROC(A1,A2,A3,A4,A5,A6,A7,A8,A9,A10,A11,A12,A13);
  /* AUDIT EXIT - CALLS APPROPRIATE SUBROUTINE BASED ON THE AUDIT */
  /* DESCRIPTOR CODE 70 - 79 */
 DCL (A1,A2,A3,A4,A5,A6,A7,A8,A9,A10,A11,A12,A13) BIN FIXED(31);
 DCL OPCODE PIC '99' BASED(P3);
 DCL (AUDEX70,AUDEX71,AUDEX72,AUDEX73,AUDEX74,
      AUDEX75,AUDEX76,AUDEX77,AUDEX78,AUDEX79) ENTRY EXTERNAL;
 DCL ENTRIES (70:79) ENTRY VARIABLE
 INIT (AUDEX70,AUDEX71,AUDEX72,AUDEX73,AUDEX74,
      AUDEX75,AUDEX76,AUDEX77,AUDEX78,AUDEX79);
 DCL PLIXOPT CHAR(50) VAR STATIC EXTERNAL
     INIT('NOCOUNT,NOFLOW,NOREPORT,NOSTAE,NOSPIE');
     P3=ADDR(A3);
     CALL ENTRIES(OPCODE) (A1,A2,A3,A4,A5,A6,A7,A8,A9,A10,A11,A12,A13);
 END;
// EXEC ????G
//G1.SYSLIB DD
//     DD
//     DD DSN=SYS1.PLIBASE,DISP=SHR
//G1.SYSIN DD *
  SYSTEM SYSID=SAMP,LOPTPARM='XREF,REUS,LET'
  GENERATE OPT=STLE,PGMID=OR,AEXIT=AUDEXIT,ALANG=PL/I
/*
```

Note: ???? is the installed ADFID (the default is MFC1).

SIGN-ON AND SIGN-OFF EXITS

These exits are for security purposes. The sign-on, or lockword, exit is provided for installation-defined checking of lockwords when users sign on to IMSADF II conversational application systems. The exit is optional; if it is not implemented, end users will not be required to enter lockwords.

The sign-off exit is called when a user terminates a session by entering OPTION C on the Primary Option Menu. It is not called if the conversation is terminated abnormally or with an IMS/VS /EXIT command (the facilities of IMS/VS must be used to achieve that - refer to the IMS/VS Installation Guide for information about abnormal conversational termination exit routines).

The sign-off exit is also called when the user chooses OPTION F (project/group switch) on the Primary Option Menu. This option permits users to switch to different application systems, if they are authorized, without having to sign on and enter a new lockword. The exit routine can provide additional control over the use of this option. (It can be disabled entirely within an application system by use of the POMENU operand of the Rules Generator GENERATE OPT=CVSYS statement.) For information on sign-on and sign-off exit processing in batch processing, refer to "Optional Lockword Exit Processing in Batch Mode" on page 10-7.

LOCKWORD EXIT

The IMSADF II Sign-On screen provides an eight-character alphanumeric field for entering a lockword. Verification of the lockword is achieved through a user-written program (Assembler, COBOL or non-main PL/I procedure) that is called by the sign-on module (MFC1TOM). Parameters that are passed to the user lockword module are:

Register	Content
1	Address of PARMLIST
14	Return address
15	Entry point address of user LOCKWORD module

The PARMLIST consists of the following:

```
    WORD 1 = ADDR (INPUT)
    WORD 2 = ADDR (ERRMSG)
    WORD 3 = ADDR (SPA)
    WORD 4 = ADDR (DBPCB)
    WORD 5 = ADDR (SRSEG)
    WORD 6 = ADDR (IOPCB)
    WORD 7 = ADDR (ALTPCB)
    WORD 8 = ADDR (EXPPCB)
    WORD 9 = ADDR (USER DBPCB LIST)
```

INPUT is an 80-byte input area from the Sign-On screen that includes the employee number, project/group, lockword, and application system ID.

Input Layout Position	Name	Length	Description
1	ILTH	2	Reserved
3	IZ1	1	Reserved
4	IZ2	1	Reserved
5	IMAN#	6	Entered employee number
11	IPJ	1	Entered project
12	IGP	1	Entered group
13	ILW	8	Entered lockword
21	IOM	8	Reserved
29	ISC	4	Application system ID from screen definition

ERRMSG is a 44-byte area used to hold any error message data that is to be displayed to the terminal user.

SPA is the scratch pad area for the active conversation. The layout of the SPA is included in IMSADF.MACLIB.ASM as member name SPAASM, SPAPLI, or SPACOBOL.

DBPCB is the first or only data base PCB that you provide. To add the PCB to the IMSADF II PSB, punch out PSB ????TOM, from library IMSADF.PSBSRC. Add the desired PCB after the Sign-On Profile PCB (DBDNAME MFDPSP01) and prior to the PSBGEN statement. Perform the PSBGEN for ????TOM. The lockword user exit routine will now have access to that PCB.

SRSEG is the SR segment retrieved from the Sign-on Profile data base.

User DBPCB LIST is the address of a list of user data base PCBs which you may use in the lockword exit, including the first but not limited to one. To add those PCBs to the ????TOM PSB, follow instructions for DBPCB above.

Note: ???? is the installed ADFID (the default is MFC1).

Parameters are returned to the sign-on module in Register 15 and, if appropriate, an error message in ERRMSG. The expected values in Register 15 are:

```
    R15 = 0   LOCKWORD OKAY
```

R15 ≠ 0   LOCKWORD ERROR, MESSAGE IN ERRMSG

The calling module's registers must be saved upon entry to the LOCKWORD module and restored to their original contents prior to returning to the caller.

If the user lockword program accesses OS/VS data sets, the data sets referenced should be identified in the IMS/VS message region job stream.

Note: IMSADF II supplies a dummy lockword exit called MFC1E01. You must replace this exit with your own exit.

- If you write your exit in COBOL or ASSEMBLER, its name must be MFC1E01. Compile the exit and link it with any subroutines you require in IMSADF.RULLIB.

- If you write your exit in PL/I, it is recommended that its name be MFC1E01.

See the _IMS Application Development Facility II Version 2 Release 2 Installation Guide_ for more information.

There are several reasons for installing a lockword exit that are considered at this time.

Multiple National Languages

If you use National Language Support processing and have more than one language installed (see the ALTLANG parameter on the DEFADF macro), you may wish to have application display screens and messages in the alternate language instead of the primary language. IMSADF II support for multilingual applications depends on your implementation of a Sign On exit to define a correspondence between a language and SYSID. Refer to the _IMS Application Development Facility II Version 2 Release 2 Installation Guide_ for more information on the ALTLANG parameter and the installation of multiple languages.

Note: It is a good idea for the implementer to define a multilingual signon screen through the Rules Generator screen image capability.

The signon module sets a new field, SPAULANG, in the SPA with the default language code for the installation and calls the lockword exit. Your code in the lockword exit module can override and set the SPAULANG value and the SPASYSID value as appropriate for the detected SYSID. If they do not correspond, the screen text and application messages will not match.

On return from the lockword exit, the Signon module validates the SPAULANG value against the languages defined at installation time in the USRLANG and ALTLANG keywords of DEFADF. It sets the common screen prefix in SPAMFSPF, which is either the first two characters of ADFID or SYSID named for the corresponding language. If the SPAULANG value is invalid, the prefix used is that of ADFID. Selection of the eight common IMSADF II screens is done using the SPAMFSPF value set at Signon.

Notes:

1. The value you set in the lockword exit is _NOT_ changed after you do a Project/Group switch (since the lockword exit is not invoked), and therefore the new application is processed in the same language as the previous one.

2. If English is used as an alternate language, then a blank (hex '40') will be used to validate SPAULANG for English rather than an 'E'. Refer to Figure 9-2 on page 9-10 for an example.

Refer to Chapter 13, "National Language Support" for additional information on National Language support.

```
   MFC1E01: PROC(INSCREN,ERRMSGAR,SPA,UDBPCB,SRSEG,IOPCB,ALTPCB,
                 EXPPCB,USRDBLST);
/***START-OF-SPECIFICATION*******************************************/
/*                                                                   */
/* MODULE NAME: MFC1E01                                              */
/*                                                                   */
/* DESCRIPTIVE NAME: MULTILINGUAL LOCKWORD EXIT                      */
/*                                                                   */
/* FUNCTION: THIS IS A SAMPLE LOCKWORD EXIT TO HANDLE MULTILINGUAL   */
/*         APPLICATIONS.  IT CHECKS SYSIDS AND SETS LANGUAGE CODES   */
/*         BASED ON THE SYSID THAT IS FOUND.                         */
/*                                                                   */
/* ENTRY POINT: MFC1E01                                              */
/*                                                                   */
/* INPUT: (PARAMETERS EXPLICITLY PASSED)                             */
/*                                                                   */
/*    SYMBOLIC NAME: SPA                                             */
/*    DESCRIPTION: CONTAINS THE SYSID AND USER LANGUAGE CODE         */
/*                                                                   */
/*    SYMBOLIC NAME: SRSEG                                           */
/*    DESCRIPTION: USER ID SEGMENT AREA                              */
/*                                                                   */
/* OUTPUT:                                                           */
/*                                                                   */
/*    SYMBOLIC NAME: SPAULANG                                        */
/*    DESCRIPTION: SET TO APPROPRIATE LANGUAGE CODE                  */
/*                                                                   */
/*    SYMBOLIC NAME: SPASYSID                                        */
/*    DESCRIPTION: SET TO ALIAS SYSID IF NECESSARY                   */
/*                                                                   */
/* EXIT NORMAL:                                                      */
/*    RETURN CODE: 0                                                 */
/*                                                                   */
/* EXIT ERROR:     NONE                                              */
/*                                                                   */
/***END-OF-SPECIFICATION*********************************************/
@INCLUDE SYSLIB(SPA);
SPAPTR = ADDR(SPA);               /* ATTACH MAP TO PARAMETER AREA    */
DCL SRSEG CHAR(44);               /* USER ID SEGMENT AREA            */
DCL 1 SYSIDTAB(18),
      2 LANGCODE CHAR(1)          /* LANGUAGE CODE PART OF VALUE     */
                 INIT('J','J','J','J','J','J',  /* JAPANESE          */
                     ' ',' ',' ',               /* ENGLISH           */
                     'F','F','F',               /* FRENCH            */
                     'G','G','G',               /* GERMAN            */
                     'P','P','P'),              /* PORTUGUESE        */
      2 THESYSID CHAR(4)          /* SYSID PART OF VALUE             */
                 INIT('KANJ','KATA','HIRA','FMIO','TATE','JAPA',
                    'MFC1','SAMP','BANK',       /* ENGLISH           */
                    'FREN','FRED','FRAN',       /* FRENCH            */
                    'GERM','KLAU','CHRI',       /* GERMAN            */
                    'PORT','LAUR','BRAZ');      /* PORTUGUESE        */
DCL ILANG FIXED BINARY(15);       /* WORK COUNTER FOR SEARCHES       */
LANGCHK:
  DO;
    DO ILANG = 1 TO DIM(SYSIDTAB); /* LOOK THROUGH ENTIRE TABLE      */
      IF SPASYSID = THESYSID(ILANG) THEN
        DO;                        /* FOUND THE SYSID                */
          SPAULANG = LANGCODE(ILANG); /* SET LANGUAGE CODE           */
          GOTO ENDLANG;            /* NO NEED FOR MORE, GET OUT      */
        END;
    END;
ENDLANG:                          /* LEAVE LANGUAGE CHECK CODE       */
 RETURN CODE(0);
 END;
```

Figure 9-2. Lockword Exit for Multilingual Applications

Non-IMSADF II Sign-On

Another way you may use the Sign on exit is to control screen flow after processing for the current application is complete. A field called SPAMODSI has the four-character MOD name of the signon screen placed in it. At conversational termination, this value plus the transaction trailer (TRXTRLTR) is inserted to the IOPCB. If you have another screen that you wish to present, write a lockword exit (or add to a current one), placing your value in SPAMODSI. There are two requirements you must meet:

1. The MFS MOD name cannot be longer than 4 characters.

2. There must be separate ones for each ADFID, since the transaction trailer is appended to the MOD name.

Bypassing SYSID Checking

The SYSID check is made after return from the lockword exit. If the SYSID field in the project group segment is blanked in your exit, no SYSID check is done. This allows SYSID checking to be bypassed under control of the lockword exit or without intervention of the lockword exit for those installations where this is necessary.

SIGN-OFF EXIT

A user exit may be invoked at sign-off to a conversation (that is, when the terminal user enters OPTION C on the Primary Option Menu or Q in most other screens). The user-written exit is invoked by the IMSADF II conversational sign-off module, MFC1T99. The exit can be written in COBOL, PL/I or Assembler. The parameters passed to the sign-off exit are:

```
    ADDR(SPA)
    ADDR(DBPCB)
    ADDR(NEWPG)
```

SPA is the scratch pad area for the active conversation. The SPA layout is included in IMSADF.ADFMAC.ASM as member name SPACOBOL, SPAPLI or SPAASM.

DBPCB is a data base PCB that you must add to the PSB ????T99. To add the PCB to the IMSADF II PSB, punch out the PSB ????T99, from library IMSADF.PSBSRC and add the new PCB just prior to the PSBGEN statement. Run the PSBGEN for ????T99 and the sign-off exit will have access to that PCB.

Note: **????** is the installed ADFID (the default is MFC1).

NEWPG is the newly signed-on project/group when OPTION F is entered on the Primary Option Menu. NEWPG is two characters in length.

When OPTION C is entered, the following information is available in the SPA:

SPACHGPG	= 0 (1 bit)
SPAPROJ	= project (1 character)
SPAGROUP	= group (1 character)
SPAMANNO	= user ID (6 characters)

When OPTION F is entered, the following information is available in the SPA:

SPACHGPG	= 1 (1 bit)
SPAPROJ	= old project (1 character)
SPAGROUP	= old group (1 character)
SPAMANNO	= user ID (6 characters)
NEWPG	= new project/group (2 characters)

Register 15 is checked upon return to the termination module, MFC1T99. A value of 0 in Register 15 indicates an OK condition. A nonzero value indicates an error condition and the error message is expected in

SPAERMSG. The Sign-On screen will be displayed with the error message in SPAERMSG.

Note: IMSADF II supplies a dummy sign-off exit called MFC1E99. You must replace this exit with your own exit.

- If you write your exit in COBOL or Assembler, its name must be MFC1E99. Compile the exit and link it with any subroutines you require in IMSADF.RULLIB.

- If you write your exit in PL/I, it is recommended that its name be MFC1E99.

See the IMS Application Development Facility II Version 2 Release 2 Installation Guide for more information.

NON-IMSADF II SIGN-OFF

A way you may use the Sign off exit is to control screen flow after processing for the current application is complete. A field called SPAMODSI has the four-character MOD name of the signon screen placed in it. At conversational termination this value plus the transaction trailer(TRXTRLR) is inserted to the IOPCB. If you have a different screen to display following signoff, write a lockword exit(or add to a current one), placing your value in SPAMODSI. There are two requirements you must meet:

1. The MFS MOD name cannot be longer than 4 characters.

2. There must be separate ones for each ADFID, since the transaction trailer is appended to the MOD name.

   You may update SPAMODSI in either sign on or sign off exit, but remember that the sign off exit takes precedence. In fact, a different screen may be requested based on transaction processing from the one named at sign on time.

DL/I EXITS

It is possible to write an exit routine that is invoked by IMSADF II every time a DL/I call is issued, whether to an application data base or to an IMSADF II data base.

The routine is invoked before and after the call and can therefore perform installation-standard editing, extra security checking, PCB switching in support of large data bases partitioned by key, and writing of audit trails.

Refer to the IMS Application Development Facility II Version 2 Release 2 Application Development Reference for details.

DIRECT USE OF THE IMS DL/I INTERFACES

The user exits (sign-on, sign-off, DL/I, and audit) as well as a special processing routine can use the IMS/VS database call interfaces ASMTDLI, CBLTDLI, and PLITDLI. Assembler and COBOL programs can use either ASMTDLI or CBLTDLI. A PL/I program can use any one of the three interfaces. A PL/I program uses ASMTDLI or CBLTDLI it must declare the interface as ASSEMBLER:

```
     DECLARE ASMTDLI ENTRY OPTIONS (ASSEMBLER);
      or
     DECLARE CBLTDLI ENTRY OPTIONS (ASSEMBLER);
```

If a PL/I program uses PLITDLI it should not be declared as ASSEMBLER:

```
DECLARE PLITDLI ENTRY;
```

Any modification of the database or PCB positioning is not known by IMSADF II and is the user's responsibility.

CHAPTER 10. BATCH PROCESSING

IMSADF II supports batch processing for entering bulk data or updating or maintaining data bases. Examples of the input format appear in Chapter 4, "The Auditor and the Audit Data Base." Batch input is useful for initial data entry and for maintaining a copy to be passed to the production system. There is no reporting or data display capability unless special processing programs are written. The output listing contains the input data itself plus confirmation or error messages.

Rules developed for use in the online system can also be applied to batch processing with only a few alterations. The same audit and message rules (but without secondary key audit or secondary transactions), the same sign-on security, and the same Rules Generator SYSTEM, SEGMENT, and FIELD statements (with extra operands on FIELD statements) can be used.

The batch driver reads input records from the TRANSIN file, acts on them by updating application system data bases and optionally writing automatic messages on the Message Data Base, and lists the input records with messages on the printer.

In an IMS/VS environment, the driver can be run as a BMP (batch message processing) program, which means that it accesses the data bases through the online IMS/VS control region instead of directly. The BMP method must be used when the data bases are online to terminal users. Only a change to the JCL is needed to switch from IMS/VS batch to BMP or vice versa.

When DB2 is your data base access method, DB2 must be invoked under the auspices of a transaction subsystem (IMS/VS or CICS/OS/VS); there are no batch attachment facilities. Figure 10-1 describes the available types of processing based on your IMSADF II installation choices.

INSTALLATION ENVIRONMENT	IMS BATCH	IMS BMP	IMS MPP	CICS TRAN
IMS/DL/I	X	X	X	
IMS/DB2		X	X	
CICS/DL/I	X			X
CICS/DB2				X

Figure 10-1. Installation Options versus Processing Environments

TRANSACTION FORMAT

The batch transaction driver is designed to behave like the online facility as far as possible; Input records are treated as transactions. An eight-character transaction code is followed by variable keys and data, the layout of which must conform with the definition given to the Rules Generator. A single batch input stream (TRANSIN) may consist of many transactions, each occupying one or more input records.

The records can be in one of two formats:

- Variable or fixed length records up to 255 bytes long. Each record begins with an 8-byte transaction code, and all the input keys and data for one transaction are present in one record.

- Fixed length 80-byte card images where one or more 80-byte records are necessary to contain the input keys and data for one transaction. In this case, a transaction can occupy up to 25 records (2,000 bytes). The number of 80-byte records required for a transaction is defined in the Rules Generator GENERATE statement for that transaction (using the CNT operand). Any individual transaction that needs fewer records than the CNT number must be terminated by end message characters defined in installation (default $$).

Each transaction begins with an eight-character transaction code of form:

ssssBmtx

where:

ssss is the application system ID

B is a literal

m is the transaction mode (1-5) with the same meaning as online. Mode 6 is allowed in special processing.

tx is the transaction ID

If an input field contains blanks, it is not mapped to the data base or pseudo segment. However, a data base field may be blanked in one of two ways:

- The input field contains a '#' in the first position followed by blanks to fill out the field, or
- The entire input field contains underscore ('_') characters.

For example, suppose that two batch transactions are defined against part of the sample data base illustrated in Figure 10-2.

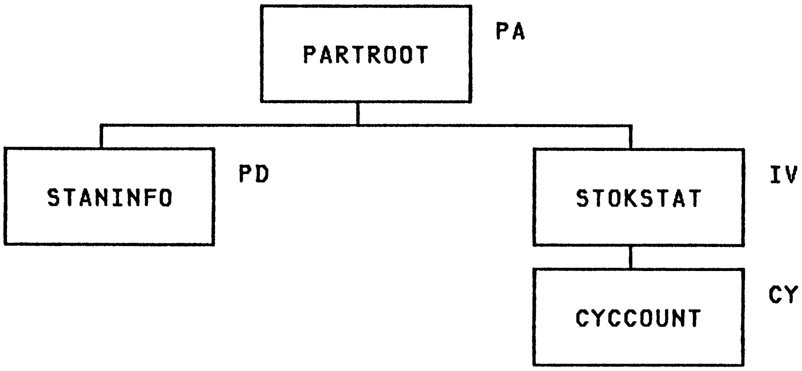


Figure 10-2. Sample Data Base

The PA transaction can update part descriptions; PD can update a four-digit field, the make department, in the dependent segment. Typical input transactions to add a new part number, alter an existing make department, and delete a part number are shown below.

```
//SAMP    JOB  ACCNT,NAME,MSGLEVEL=1
//BATCH   EXEC DLIBATCH,MBR=SAMPBDPP,RGN=728K
//DI21PART DD DSN=IMSVS.DI21PART(PRIME),DISP=OLD
//DI21PARO DD DSN=IMSVS.DI21PARO,DISP=OLD
//PRINTER  DD SYSOUT=A
//RSTRTIN  DD DUMMY
//TRANSIN  DD *
SAMPB4PA02X4CV76BQE        LEFT HANDED WIDGET
SAMPB5PD03B276             021728
SAMPB3PA47C37X26
/*
```

Figure 10-3. JCL for Batch Processing

In this example, the program being executed is a version of the batch transaction driver especially tailored by the Rules Generator process the PA and PD transactions in the SAMP system. The next section explains how to write rules to tailor the batch transaction driver.

Typical output from the above input is shown in Figure 10-4.

```
      PAGE 00001             SAMPLE DATA BASE         04/05/79
TIME=16:38:328

          RECORD # 00001   SAMPB4PA02X4CV76BQE      LEFT HANDED WIDGET
     TRANSACTION # 00001   *** SEGMENT ADDED SUCCESSFULLY ***

          RECORD # 00002   SAMPB5PD03B276             021728
     TRANSACTION # 00002   9325 MAKE DEPARTMENT OUTSIDE ALLOWED RANGE

          RECORD # 00003   SAMPB3PA47C37X26
     TRANSACTION # 00003   *** SEGMENT DELETED SUCCESSFULLY ***

                           *** END OF INPUT ***
```

Figure 10-4. Typical Output from Batch Processing

The second transaction was not completed because of an error detected by audit rules.

ERROR HANDLING

If an auditing, data base, or other error occurs during batch processing, the batch transaction driver sets a nonzero return code which can be checked using the condition code facility (COND) of JCL. This does not apply to a BMP.

To make it easier for the user of a batch system to locate and correct transactions in error, the batch driver JCL can be supplemented with two optional DD cards:

- ERRTRX will contain a copy of those input transactions that were found to be in error. After they are corrected, the transactions in this file can be resubmitted.
- ERRMSG will contain a listing of the transactions in error with the associated error messages.

RULES

As for online conversational processing, the Rules Generator expects SYSTEM, SEGMENT, FIELD and GENERATE statements. SYSTEM and SEGMENT statements can be used unaltered to define batch applications. FIELD statements require the additional operands shown below.

FLDPOS The starting position of the field in any batch transaction in which the field is used. The first available position is 9 (11 for variable length TRANSIN records). If more than one card image makes up a transaction, the first position in the second card is 81 and so on. FLDPOS is used in conjunction with the DISPLAY operand value to provide compatibility with online conversational transactions using the default screen layouts (SPOS=AUTO). When generating a batch transaction you will include only those fields that would be displayed in the equivalent online conversational transaction screen. This means that the field is in the batch transaction only if FLDPOS is specified and DISPLAY=YES is either specified or implied by default.

ILENGTH The input length of the field, if this differs from the LENGTH operand value.

ITYPE The input data type. The default is alphanumeric. Allowed values are the same as those for the TYPE operand. Data conversions will be performed.

These operands are ignored when generating rules for online conversational applications; therefore, the same definitions can be used to create a batch system that is equivalent to the online system and that can be used as a backup.

As for conversational processing, GENERATE statements for segments and transactions are required. Those for segments are the same and need not be repeated. Those for transactions differ from their online conversational counterparts as follows:

OPTIONS Specify OPT=BAIT (Batch Input Transaction Rule) for a batch transaction.

CNT The number of input records making up one transaction. The default is 1, allowing a fixed or variable record format of up to 255 bytes in length. If CNT is greater than 1, the input records must be 80 bytes and of fixed length. The CNT maximum is 25.

Various activities for conversational processing, such as building menus, are unnecessary in batch processing. However, a link edit must be performed to build a tailored transaction driver able to process a particular set of batch transactions. The result of the link edit is a program that can be invoked using the JCL shown in the example above. The Segment Handler Rules required during the execution of the driver are included in the link edit. Consequently, the batch driver link edit request should be the last GENERATE statement in the Rules Generator input. The following operands on the GENERATE statement are relevant to link editing a batch driver:

OPTIONS Specify BDLE (batch driver link edit).

PGMID A two-character ID that determines the name of the resultant executable program, which has the form:

xxxxBDID

where:

ssss is the application system ID
BD is a literal
ID is the value of the PGMID operand

SHTABLE List of all data base segments (DBPATH including all higher levels and TSEGS) used in all transactions to be processed by this batch transaction driver program. Must be complete but must not name pseudo segments.

ITTABLE Complete list of standard processing transaction IDs to be processed by this batch transaction driver program.

PHEADING The heading to appear on the transaction listing at execution time. May be up to 60 characters and must be enclosed in quotation marks. Optional.

WTOMSG A message to be written to the system operator at start of execution. May be up to 60 characters and must be enclosed in quotation marks. The operand may be coded twice to send a longer message. Optional.

WTORMSG A message that invites the system operator to stop the program before it has finished by replying STOP. May be up to 60 characters and must be enclosed in quotation marks. If he replies STOP, the driver will complete the current transaction, copy the remaining input records to a file (TRANSOUT), and terminate. The saved file may then be used as input (TRANSIN) to a later run that completes the work. The operand may be coded twice to send a longer message. Optional.

SIGNON A value of YES stipulates that the first input record to any execution of this batch transaction driver program must be a valid sign-on. (Discussed below.)

CHKPT A value of YES requests checkpointing for this batch driver execution. (Discussed below.)

FREQ Specifies the number of valid (syntactically correct) transactions to be processed between checkpoints.

Example

Here are the Rules Generator statements that define the batch application shown in the previous example:

```
SEGMENT PARENT=0,ID=PA,NAME=PARTROOT,LENGTH=50
  FIELD ID=KEY,LENGTH=17,KEY=YES,NAME=PARTKEY,FLDPOS=9
  FIELD ID=DESC,LENGTH=20,POS=27,FLDPOS=26
SEGMENT ID=PD,NAME=STANINFO,LENGTH=85,PARENT=PA
  FIELD ID=KEY,LENGTH=2,KEY=YES,NAME=STANKEY,FLDPOS=26
  FIELD ID=MKDP,LENGTH=4,POS=48,FLDPOS=28
GENERATE OPT=SGALL
GENERATE OPT=BAIT,TRXID=PA,DBPATH=PA
GENERATE OPT=BAIT,TRXID=PD,DBPATH=PD
GENERATE OPT=BDLE,PGMID=PP,SHTABLE=(PA,PD),ITTABLE=(PA,PD)
```

Note: The GENERATE statements for the segments are shown although these are not necessary if the segment rules have previously been generated for online use.

CREATING OUTPUT AND REPORTS

When the batch transaction driver executes as a BMP, any secondary transactions and IMS/VS messages are sent via IMS/VS. In a batch region, however, the messages are written to an output data set with DD name SECTRX. It is thus possible to create output and reports under the control of the high level audit language, using the SEND IMMED 'ssORxx01' statement described in Chapter 7, "Secondary Transactions and IMS/VS Message Routing." An example of this facility is given in Appendix C, "Report Writing Example."

PAGE AND SPACE CONTROL

The output listing, from the batch execution, prints all transaction input plus any error, warning or informational messages. The spacing in this report can be controlled by statements placed in the input stream. These control statements are:

EJECT Causes the page to eject after the statement is encountered. The format is: EJECT (column 2-6).

SKIP nn Causes the report to skip after the statement is encountered. The number specified in nn can be 1 to 99. The format is: SKIP nn (column 2-5 and 7-8).

MESSAGE CONTROL

The messages generated during a batch execution can be expanded through the insertion of a Message Control statement in the input stream. This statement causes additional information to be mapped into some of the error messages. The format is:

MSG=(n,m)

where:

MSG indicates that this is a Message Control statement (columns 2-4)

n specifies option 1 (0, 1, or 2)

m specifies option 2 (1 or 2)

The statement can be coded with either the n option, or n and m. The allowable parameters are:

0 no additional expansion of the message. This is the default and is the format if message control is not used.

1 specifies that data base error messages (not found and already exists) will have the segments keys mapped into the message.

2 specifies that error and warning message will be expanded to show the name of the field in error and its contents.

SIGN-ON SECURITY

If SIGNON=YES is coded on the GENERATE OPT=BDLE statement, a SIGNON transaction must precede any other transaction during input. The format is:

SIGNON pg userid lockword

where:

SIGNON indicates that this is a sign-on transaction (columns 1-6)

pg is the project/group (columns 8-9)

userid is the user ID (columns 11-16)

lockword is optional: if specified, you must supply a lockword exit (columns 18-25)

For example:

SIGNON YY 999999

IMSADF II makes sure that the user ID and project/group are permitted to use the application system ID of the batch transaction driver program and restricts the use of transactions according to the security profile.

There is also a SIGNOFF transaction (coded in columns 1-7). This is useful when batch input to a run is merged from several sources. Each submitter of transaction input should have a SIGNON at the start and a SIGNOFF at the end to prevent the next submitter running under the first sign-on.

If any errors are detected during SIGNON processing, an error message is printed and the transactions are flushed until a valid sign-on is encountered.

OPTIONAL LOCKWORD EXIT PROCESSING IN BATCH MODE

The submitter may be required to include a lockword in the SIGNON transaction record. Verification is achieved through an exit routine in Assembler, PL/I, or COBOL that is called by the batch transaction driver. The parameters that are passed to the lockword exit are the same in batch as in conversational processing. Therefore, the discussion of lockword processing in Chapter 9, "Exits" applies to batch processing with the following exceptions:

For PL/I lockword exits:

- To incorporate the lockword exit routine with the batch driver, the batch driver rule must be relinked, including the appropriate link-edit CHANGE statement.

- To incorporate a non-main PL/I procedure as an exit, the following Linkage Editor control statements are required:

```
    INCLUDE   OBJLIB(LOCKUPGM)
    CHANGE    MFC1E01(LOCKUPGM)
    INCLUDE   OBJLIB(MFC1E01P)
    CHANGE    MFC1E01C(MFC1E01P)
    INCLUDE   SYSLMOD(ssssBDxx)
    ENTRY     MFC1T09
    NAME      ssssBDxx(R)
```

  where:

  xx is the batch driver rule ID
  ssss is the application system ID

If your sign-on exit was written on COBOL or Assembler, it was incorporated automatically by SMP in load module ????BXXX. Since that module is included in a batch transaction driver link-edit, the exit now exists in your batch transaction drivers under the name MFC1E01.

SIGN-OFF EXIT IN BATCH MODE (OPTIONAL)

An exit routine can be written to be invoked at sign-off time in batch processing if sign-on is used. "Sign-off time" occurs when a SIGNOFF or another SIGNON transaction is encountered, or at end-of-file on TRANSIN.

The parameters passed to the sign-off exit are the same as in conversational processing, with the following exceptions.

- The parameter ADDR(NEWPG) is set to zero since it does not apply to batch. The information available in the SPA is the same that is available when OPTION C is entered in conversational mode.

For PL/I lockword exits:

- To include a sign-off exit with the batch driver, the batch driver rule must be relinked, including the appropriate link-edit CHANGE statement.

- To incorporate a non-main PL/I procedure as an exit, the following linkage editor control statements are required:

```
    INCLUDE   OBJLIB(signoff exit)
    CHANGE    MFC1E99(signoff exit)
    INCLUDE   OBJLIB(MFC1E99P)
    CHANGE    MFC1E99C(MFC1E99P)
    INCLUDE   SYSLMOD(ssssBDxx)
    ENTRY     MFC1T09
    NAME      ssssBDxx(R)
```

where:

xx is the batch driver rule ID
ssss is the application system ID

If your sign-off exit was written on COBOL or Assembler, it was incorporated automatically by SMP in load module ????BXX. Since that module is included in a batch transaction driver link-edit, the exit now exists in your batch transaction drivers under the name MFC1E99.

CHECKPOINTS

Checkpoints will be taken at intervals during batch processing if CHKPT=YES is specified on OPT=BDLE statement to the Rules Generator. Then, each time the batch transaction driver program is executed, checkpoints will be taken under the following conditions:

- CHKPT transaction (consisting of the word CHKPT in columns 1 to 5) included in the input stream. This may occur at any time during processing, but it must not occur within a data base transaction.
- A predefined frequency of input data base transactions is met. This frequency is specified in the FREQ=nnnnn operand of the GENERATE statement.
- A combination of the first two conditions.

RESTART PROCESSING

Restart processing may be performed with the batch transaction driver. Data base backout, however, is not performed. The IMS/VS data base backout utility must be performed for any DL/I data base updates after the last checkpoint prior to restarting the job.

Restart processing may be triggered by a data set with DDNAME RSTRTIN. The record format of this data set is fixed length, 80 characters. The format of the command is:

RSTRT CHKPTID ABENDNO

where:

RSTRT is the command (columns 1-5)

CHKPTID is the ID of the checkpoint at which restart is to begin (columns 7-14).

ABENDNO if specified, indicates the starting input record number of a transaction which is to be skipped (possibly a transaction causing a previous abend) (columns 16-20). **Optional.**

The format is nnnnn. This number will appear on the batch output listing when the input record is printed.

In an IMS/VS environment, you may alternatively trigger restart processing by specifying the checkpoint ID in the EXEC PARM statement. See the _IMS/VS Application Programming Reference Manual_ and _IMS/VS System Programming Reference Manual_ for further information.

SPECIAL PROCESSING

Both standard and special processing are supported in batch. A Special Processing Routine (SPR) receives control in the same manner as during online processing and is passed a communication work area having exactly the same layout as the SPA. The SPR can call subroutine services such as the MAPPER and SEGUPDTE in the same way. All the calls except DISPLAY are available to it. When the SPR terminates, it passes a return code to the batch transaction driver to tell it what to do. The return codes are equivalent to those for online processing where possible. The batch transaction driver performs automatic message sending on return from the SPR when appropriate rules are present.

BYPASS Specify BYPASS=YES for a special processing Batch Input Transaction Rule (GENERATE OPT=BAIT) when the batch transaction driver is to make two calls to the special processing routine. The first call is made after the DBPATH segments have been returned and the second is made after the input transaction data mapper has mapped the changed field into the SPA work area for batch processing. The default is NO.

Certain special processing routines open and use their own DCBs. These DCBs can be closed by the special processing routines through a final maintenance call that is made available when EOF=Y is coded on the GENERATE OPT=BAIT statement. For this maintenance call, the batch transaction driver calls the special processing routine when an end-of-file condition is encountered on the TRANSIN data set. The only parameter passed to the special processing routine is the address of the SPA. The SPAFIRST field in the SPA will contain the value -1.

Batch special processing uses the following operands on the GENERATE OPT=BDLE statement:

SPTABLE A list of all special processing transactions to be processed by this batch transaction driver program. The Rules Generator will link-edit the programs with the batch transaction driver. They must all be present with correct names (of form ssssUtx, where tx is the transaction ID) in the OBJLIB data set, which must be available to the Rules Generator.

SHTABLE The list should include all segment IDs referenced in SEGHNDLR calls by all programs identified in SPTABLE.

MAPTABLE Mapping segments used in the special processing routines should be listed.

RETURN CODES

Code	Transaction Driver Action
0,2	Read next transaction.
1	Generate secondary transactions as required according to the setting of SPASECTX. Return control immediately to the SPR.
3	Output the message in SPAERMSG and read next transaction.
4	Output **SPECIAL PROCESSING COMPLETED SUCCESSFULLY** and read next transaction.
8	Output the audit error message and read next transaction.
12	SPR has written the error message already. Read next transaction.
24	Issue ROLL call to back out any updates. Pseudo abend caused.
28	Generate and print error message returned from the segment handler. Read next transaction.
32	Read next transaction.
XX	Invalid return code (XX=other). Message written; batch transaction driver sends return code of 44.

Notes:

1. The error message is specified by a message number in the COMSG field of the segment handler communication area.

2. Return code 100 is an internal return code meaning that the special processing routine could not be found. **DO NOT USE THIS VALUE.**

BATCH APPLICATION IMPLEMENTATION CHECKLIST

Please refer to Figure 10-1 on page 10-1 for environment-related information.

1. Generate any special processing routines to handle complex logic according to the specifications provided in "Special Processing" on page 10-9. Generate the special processing interface data module table, MAPTABLE, to indicate mapping requirements, via the GENERATE statement with OPTIONS=BDLE and MAPTABLE=mapids.

2. Generate rules and data bases to define transaction message processing. The rules to include are:

   - Input Transaction Rules (OPTION=BAIT)
   - Segment Handler and Segment Layout Rules (OPTION=SGALL)
   - Batch Driver Rule (OPTION=BDLE)

3. If auditing and automatic message sending are required, generate appropriate information in the Audit and Message Data Bases.

4. Link-edit the batch transaction driver with the Batch Driver Rule and, if special processing is used, with the special processing routines, the special processing interface routine (SPIR), and the special processing interface data module. The GENERATE statement with OPTIONS=BDLE causes the Rules Generator to perform the link-edit.

5. Generate a PSB containing the PCBs for DL/I data base access.

6. Prepare a job stream for batch execution (if appropriate).

7. Prepare the TRANSIN sequential data set to contain transaction records to be processed.

8. Add the APPLCTN macro to the IMS/VS system definition if the program is to be executed as a BMP program.

9. For CICS/OS/VS, update the following tables as appropriate:

   - DFHPPT
   - DFHPSB (DL/I only)
   - DFHDBD (DL/I only)

   Note: See the _IMS Application Development Facility II Version 2 Release 2 Installation Guide_ for DFHDCT entries required for execution of a batch driver in a CICS/OS/VS environment.

BATCH DRIVER COMPLETION CODES

Condition codes returned by the batch transaction driver are listed below. The actual code returned during a batch run will be the highest value encountered during processing. In most cases, a message describing the problem will follow the transaction or record in error. Exceptions are noted. Condition code values of 60 or less indicate that processing continued. Condition codes greater than 80 indicate that processing was terminated when the error condition was encountered.

Code	Explanation
0	Successful processing - no errors encountered
4	Special processing routine wrote or set up a message (i.e., returned code 3, 12 or 28). Message could be a successful completion, an error message, or no more checkpoints allowed (see message).
8	Syntax error in input record
12	Audit error
16	Data mapper error
20	DL/I error
24	Input Transaction Rule or other rule not found (see message)
40	Soft Stop - operator keyed in STOP (no message given)
44	Unknown return code from special processing routine
60	Sign-on/sign-off error encountered
76	End of data reached on TRANSIN before restart complete
80	Error occurred during checkpoint
84	Rule(s) not in library, message indicates which
88	DCB characteristics of TRANSIN unacceptable
92	Error encountered during restart
96	TRANSIN, TRANSOUT, or RSTRTIN could not be opened, message indicates which
100	PRINTER could not be opened (no message given)

CHAPTER 11. NONCONVERSATIONAL PROCESSING

Under CICS/OS/VS, the nonconversational display screen is obtained by entering:

tttt xxxxxxxx

where,

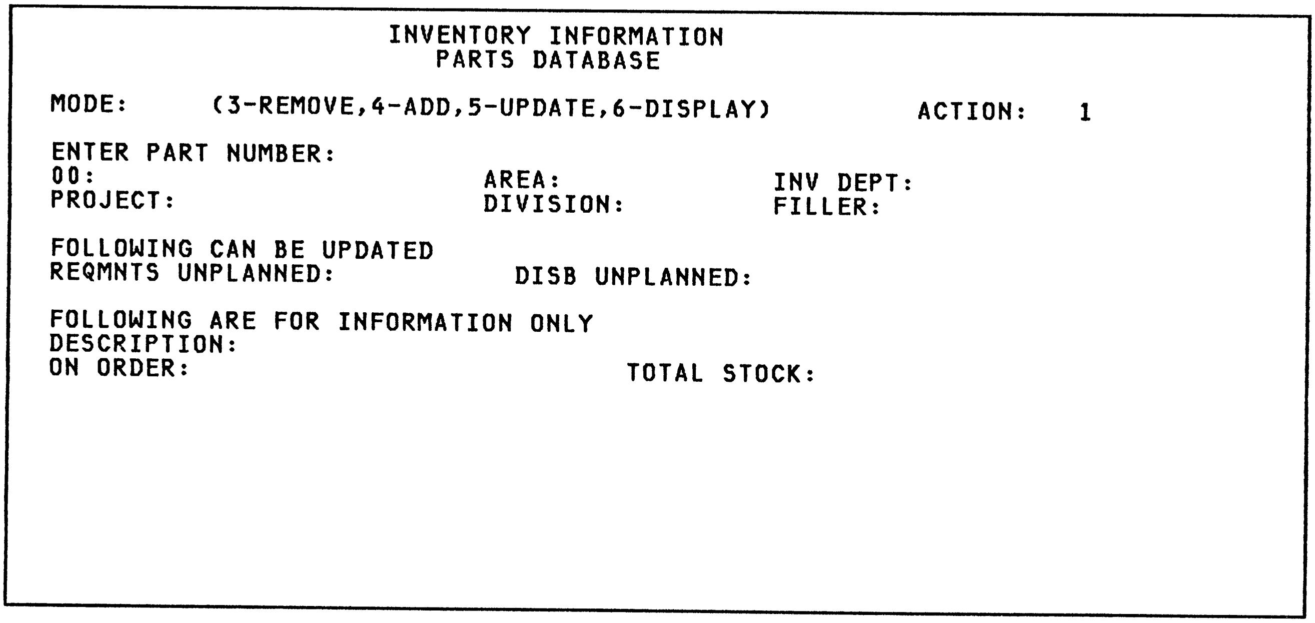
tttt

> is a one- to four-character transaction code defined to CICS/OS/VS which will give control to IMSADF II.

xxxxxxxx

> is the one- to eight-character Output Format Rule name for the desired transaction. (See the MODNAME operand of the GENERATE statement in the IMS Application Development Facility II Version 2 Release 2 Application Development Reference.)

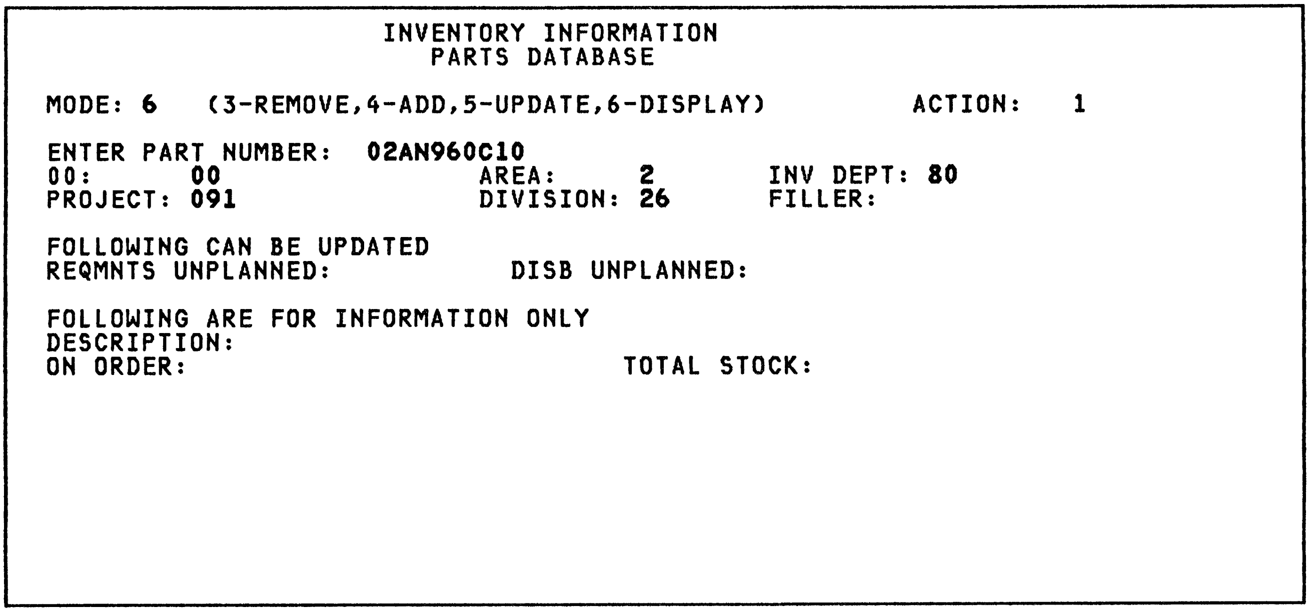
Under IMS/VS, the nonconversational display screen is obtained by entering the IMS/VS /FORMAT command. Enter /FOR xxxxxxxx and press the ENTER key to display the desired screen, where xxxxxxxx is the appropriate IMS/VS MFS modname.



```
                    INVENTORY INFORMATION
                       PARTS DATABASE

MODE:     (3-REMOVE,4-ADD,5-UPDATE,6-DISPLAY)        ACTION:   1

ENTER PART NUMBER:
00:                          AREA:              INV DEPT:
PROJECT:                     DIVISION:          FILLER:

FOLLOWING CAN BE UPDATED
REQMNTS UNPLANNED:             DISB UNPLANNED:

FOLLOWING ARE FOR INFORMATION ONLY
DESCRIPTION:
ON ORDER:                              TOTAL STOCK:
```

Figure 11-1. Nonconversational Display Screen

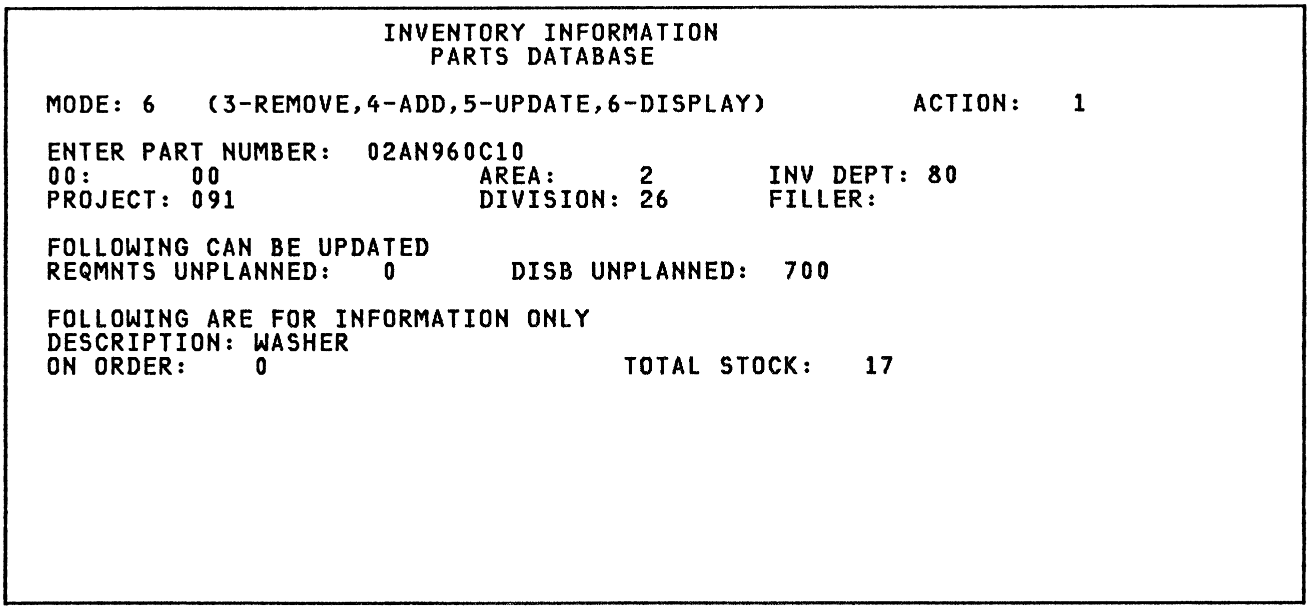
The user must now select one of the transaction modes and enter key information, as in Figure 11-2.



```
                        INVENTORY INFORMATION
                           PARTS DATABASE

  MODE: 6   (3-REMOVE,4-ADD,5-UPDATE,6-DISPLAY)        ACTION:   1

  ENTER PART NUMBER:  02AN960C10
  00:       00                AREA:     2      INV DEPT: 80
  PROJECT: 091                DIVISION: 26     FILLER:

  FOLLOWING CAN BE UPDATED
  REQMNTS UNPLANNED:            DISB UNPLANNED:

  FOLLOWING ARE FOR INFORMATION ONLY
  DESCRIPTION:
  ON ORDER:                            TOTAL STOCK:
```

Figure 11-2. Requesting a Display by Entering Key Information

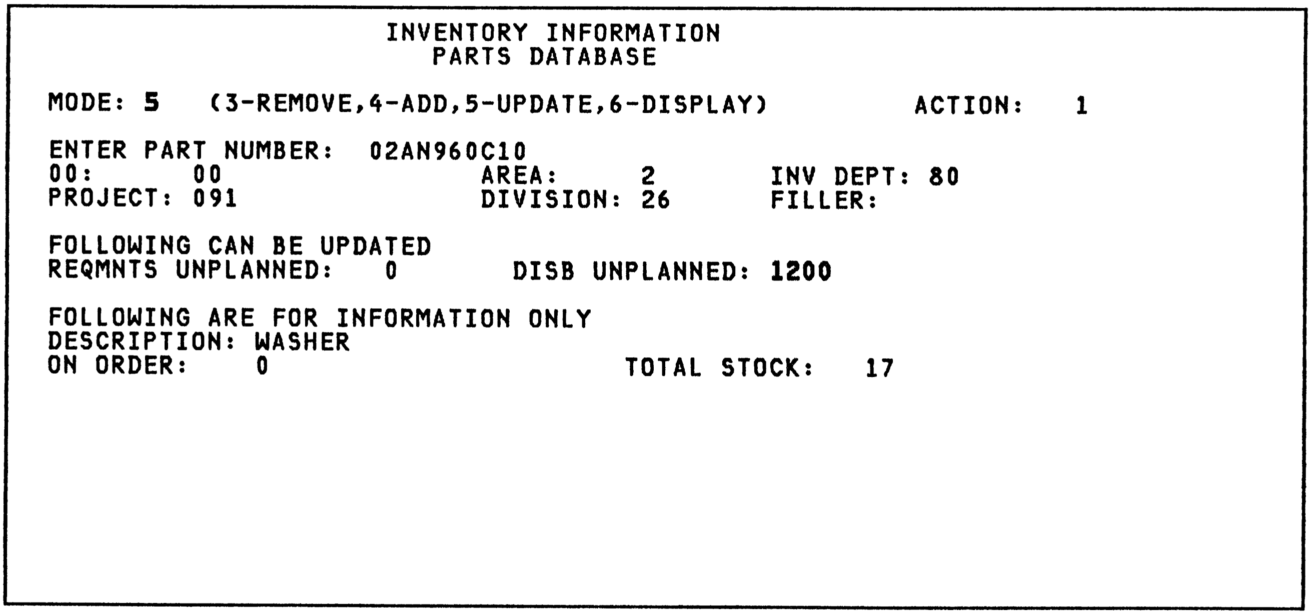
If mode 6 (Display) is entered, data will be displayed on a screen like the one in Figure 11-3.



```
                        INVENTORY INFORMATION
                           PARTS DATABASE

  MODE: 6   (3-REMOVE,4-ADD,5-UPDATE,6-DISPLAY)        ACTION:   1

  ENTER PART NUMBER:  02AN960C10
  00:       00                AREA:     2      INV DEPT: 80
  PROJECT: 091                DIVISION: 26     FILLER:

  FOLLOWING CAN BE UPDATED
  REQMNTS UNPLANNED:   0        DISB UNPLANNED:  700

  FOLLOWING ARE FOR INFORMATION ONLY
  DESCRIPTION: WASHER
  ON ORDER:    0                       TOTAL STOCK:   17
```

Figure 11-3. Data Display

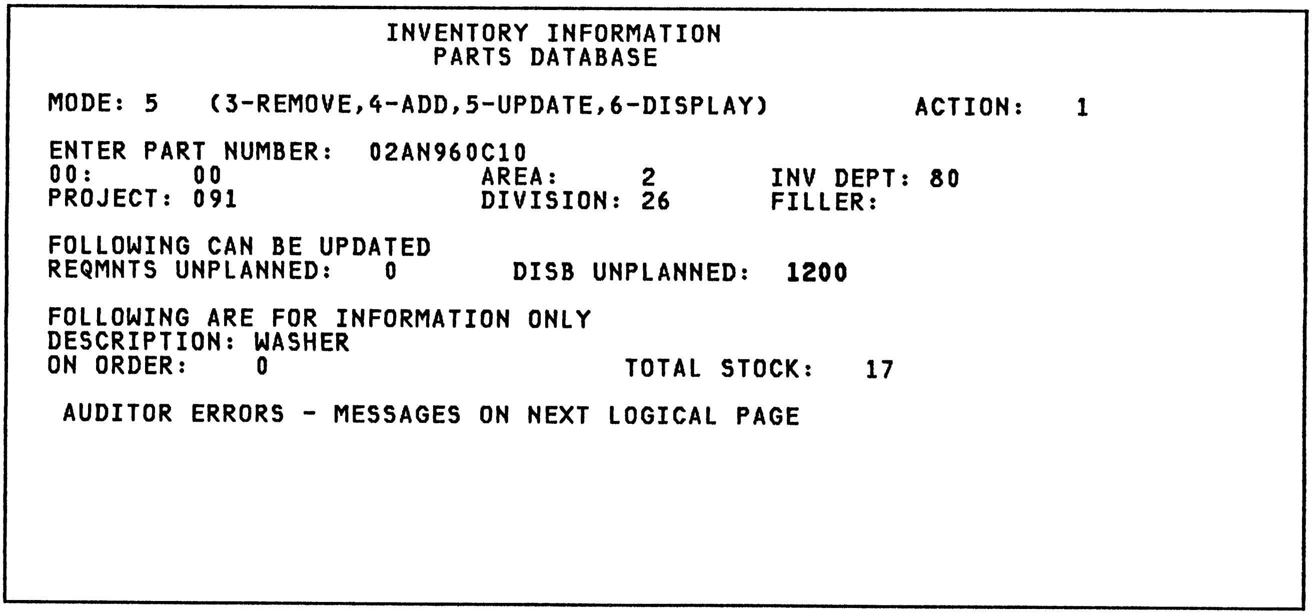
If the user changes the mode to 5, he can amend the data (as shown in Figure 11-4), and the data base will be updated.



```
                    INVENTORY INFORMATION
                       PARTS DATABASE

MODE: 5   (3-REMOVE,4-ADD,5-UPDATE,6-DISPLAY)        ACTION:   1

ENTER PART NUMBER:  02AN960C10
00:       00                AREA:     2      INV DEPT: 80
PROJECT: 091                DIVISION: 26     FILLER:

FOLLOWING CAN BE UPDATED
REQMNTS UNPLANNED:   0       DISB UNPLANNED: 1200

FOLLOWING ARE FOR INFORMATION ONLY
DESCRIPTION: WASHER
ON ORDER:    0                       TOTAL STOCK:   17
```

Figure 11-4. Making an Amendment

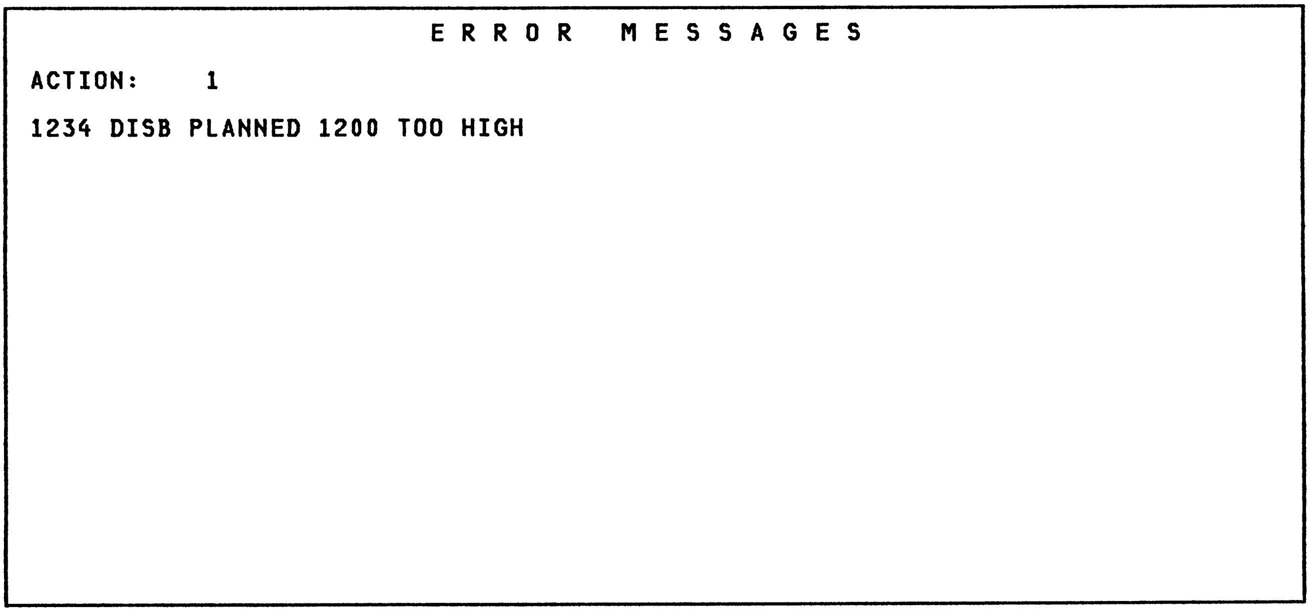
When you define the rules for this format, certain fields may be defined as display only, while others may be subject to validation and other processing before the data base will be updated. These rules are interpreted by the Auditor, a part of the transaction driver. If errors are detected during these validation checks, the user is informed, as shown in Figure 11-5; the field in error is highlighted.



```
                    INVENTORY INFORMATION
                       PARTS DATABASE

MODE: 5   (3-REMOVE,4-ADD,5-UPDATE,6-DISPLAY)        ACTION:   1

ENTER PART NUMBER:  02AN960C10
00:       00                AREA:     2      INV DEPT: 80
PROJECT: 091                DIVISION: 26     FILLER:

FOLLOWING CAN BE UPDATED
REQMNTS UNPLANNED:   0       DISB UNPLANNED:  1200

FOLLOWING ARE FOR INFORMATION ONLY
DESCRIPTION: WASHER
ON ORDER:    0                       TOTAL STOCK:   17

 AUDITOR ERRORS - MESSAGES ON NEXT LOGICAL PAGE
```

Figure 11-5. A Validation Error Detected by the Auditor

If the user needs more information to correct the error, he presses PF key 1, or types =+1 in the ACTION field and presses ENTER to display error messages. Figure 11-6 shows an example.



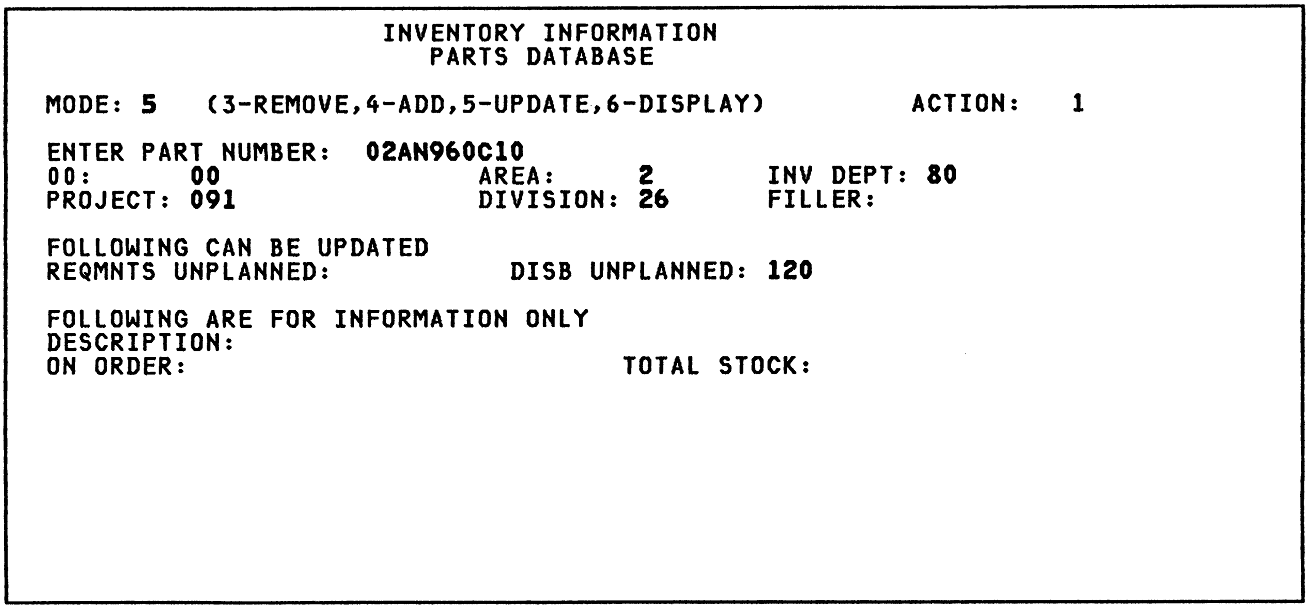
```
                         E R R O R   M E S S A G E S
 ACTION:    1

 1234 DISB PLANNED 1200 TOO HIGH
```

Figure 11-6. Error Message

To complete the updates, the user presses PF key 2 or types =-1 in the ACTION field and presses ENTER. The display screen will return.

ACTION: =+1 (PF key 1) means show the next logical page; ACTION: =-1 (PF key 2) means show the previous logical page; ACTION: =1 (PF key 3) means show the first logical page, which is the one containing the data.

It is not necessary to display data before updating it. On the first display screen, the user can enter mode 5, the key fields, and the amendments, as shown in Figure 11-7.

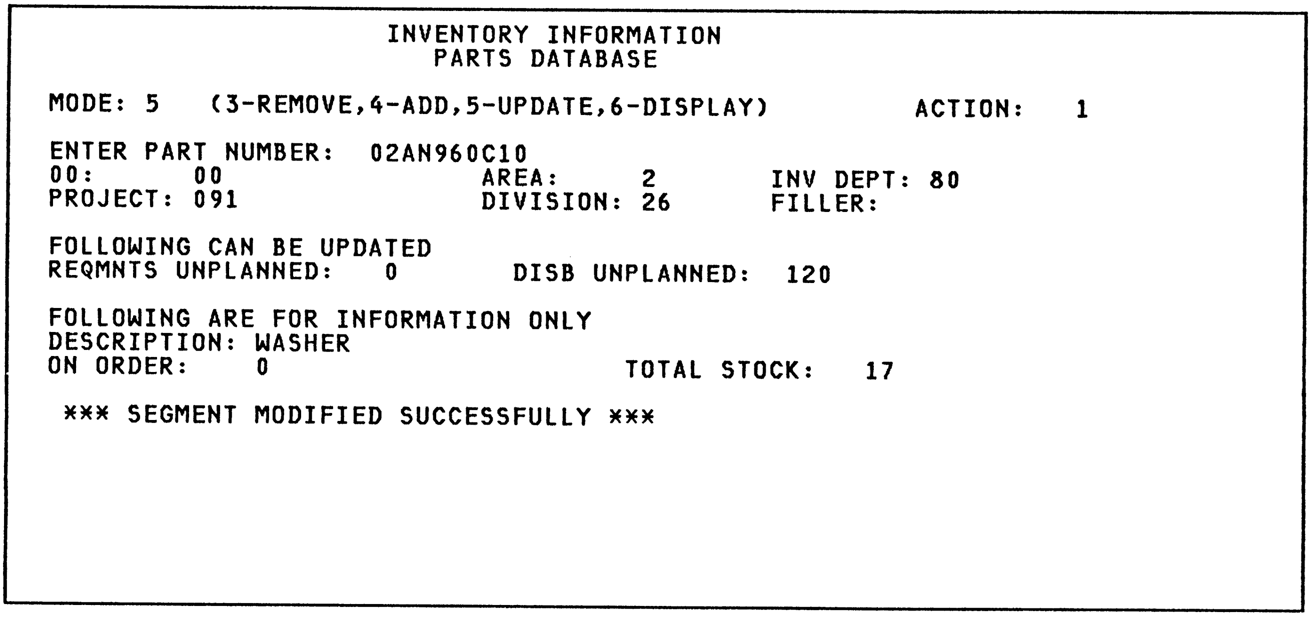


```
                         INVENTORY INFORMATION
                            PARTS DATABASE

  MODE: 5   (3-REMOVE,4-ADD,5-UPDATE,6-DISPLAY)        ACTION:   1

  ENTER PART NUMBER:  02AN960C10
  00:       00                AREA:      2       INV DEPT: 80
  PROJECT: 091                DIVISION: 26       FILLER:

  FOLLOWING CAN BE UPDATED
  REQMNTS UNPLANNED:            DISB UNPLANNED: 120

  FOLLOWING ARE FOR INFORMATION ONLY
  DESCRIPTION:
  ON ORDER:                                TOTAL STOCK:
```

Figure 11-7. Entering an Amendment Without First Requesting a Display

Again, if errors are detected, the user will be informed as shown in Figure 11-5 and Figure 11-6.

When updates are successful, final confirmation will appear, as shown in Figure 11-8.



```
                         INVENTORY INFORMATION
                            PARTS DATABASE

 MODE: 5    (3-REMOVE,4-ADD,5-UPDATE,6-DISPLAY)         ACTION:   1

 ENTER PART NUMBER:   02AN960C10
 00:        00                AREA:     2        INV DEPT: 80
 PROJECT: 091                 DIVISION: 26       FILLER:

 FOLLOWING CAN BE UPDATED
 REQMNTS UNPLANNED:    0        DISB UNPLANNED:   120

 FOLLOWING ARE FOR INFORMATION ONLY
 DESCRIPTION: WASHER
 ON ORDER:     0                      TOTAL STOCK:    17

  *** SEGMENT MODIFIED SUCCESSFULLY ***
```

Figure 11-8. Confirmation Message

This sample transaction works against the data base shown in Figure 11-9. Each segment has a two-character ID. The example given permits display and update of the PA and IV segments; the key fields the user is required to enter are the keys of these segments.

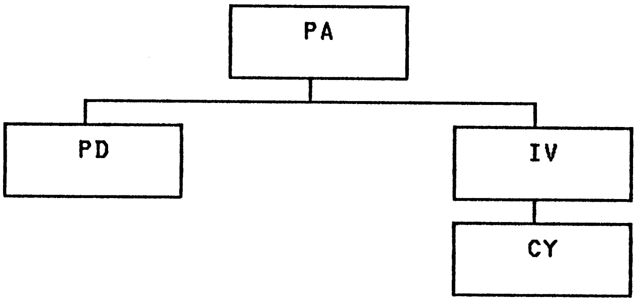


Figure 11-9. Data Base Used in Examples

The lowest level segment retrieved by a transaction in a hierarchical path is known as the **target segment**. When a transaction involves multiple hierarchical paths in one or more data bases, it is said to have multiple target segments - one for each path. When defining a transaction, you will name the target segments.

Transaction modes 3 and 4 treat target segments differently from other segments. Mode 4 will insert a new occurrence of the target in the data base. If mode 4 were used in the example, the user would enter an existing part number and a new inventory key. He would enter data to go into the inserted segment at the same time. Again, the Auditor would verify the data, and a confirmation display like the one in Figure 11-8 would appear, with the message ***** SEGMENT ADDED SUCCESSFULLY *****.

Transaction mode 3 is used for deleting segments. However, it is used mainly on transactions with a single target segment. Thus, if used in the sample transaction, mode 3 would delete the IV segment. A multiple-path transaction may also delete segments. However, such a function is under control of audit rules (rules in the Audit Data Base) and is not restricted to mode 3.

The example above illustrates a single transaction with a single target segment. The transaction has a two-character ID; this ID is hidden from the terminal user.

For single path transactions, a simple convention is that transaction ID equals target segment ID. You can then define a set of data base maintenance transactions, one for each segment type. These can later be supplemented by more complex transactions.

Therefore, to perform data base maintenance (including test data creation) against the sample data base, transactions PA, PD, IV and CY would be defined.

Transactions are collected together into application systems. An application system is given a four-character ID. Like the transaction ID, this ID is not visible to the user, but it must be assigned. The ID for the sample application system used here is SAMP.

STATIC RULES AND THE RULES GENERATOR

An IMSADF II transaction consists of a generalized application program controlled by rules. Static rules are used to define:

- The transactions within an application system
- The data base and its segments
- Which segments will be used in each transaction
- what data is to appear on the display screens
- The audit specifications and interrelationships that will be required in a transaction

These rules are "static" because they are relatively stable and unchanging. The IMSADF II Rules Generator, a utility like a compiler, processes static rules and stores them as members in an OS partitioned data set (PDS).

The next section describes the various types of static rules. The statements you will use to submit these rules and other instructions to the Rules Generator will be introduced in the following section.

IMSADF II retrieves the rules required according to the user's sign-on and subsequent selection of transactions.

STATIC RULES FOR NONCONVERSATIONAL APPLICATION SYSTEMS

The following static rules are required for every transaction:

Input Transaction Rule	Defines the segments to be used in a transaction, including a small amount of information about the kind of processing to be performed against the data base.
Output Format Rule	Tells the transaction driver what fields to display.

The following static rules are required for every data base segment to be used:

Segment Handler Rule
: Contains the actual segment search arguments (SSAs) that IMSADF II needs to perform data base I/O using DL/I. One is produced for every data base segment to be used.

Segment Layout Rule
: Defines the fields in a segment, including their length and format, and indicates whether any validation or message sending is to be performed.

Table Handler Rule
: Builds an Assembler program containing standard static SQL calls and USER SQL calls.

Table Layout Rule
: Defines the columns in a DB2 table. It performs the same function as the Segment Layout Rule.

There are six types of source statements to be submitted to the Rules Generator. They are:

SYSTEM Defines the application system ID and sets general system parameters.

SEGMENT Defines the data base layout (similar to an IMS/VS DBD); segments are usually defined in hierarchical order. (Data bases are defined implicitly through the PCBs.) There must be a separate SEGMENT statement for every data base segment used in a transaction.

FIELD Defines the key and data fields contained in a segment and indicates how the fields are to be displayed on the screens in which they appear.

GENERATE Has several uses:

- Defines transactions, controls screen formats, and identifies which data base segments are to be used
- Controls the generation of Segment Handler and Segment Layout Rules (using information given in the SEGMENT and FIELD statements)
- Requests link edit and preload performance options

RULE Provides control information to the Rules Generator for entering Assembler language rules source.

Note: The RULE statement is not supported under the Interactive Application Development Facility (IADF).

INCLUDE Provides a copy library facility that allows basic information to be stored, retrieved, and augmented or overridden by additional statements and parameters.

Figure 11-10 shows how these Rules Generator statements can be used to produce a single transaction, PA, which allows display and update (including insertion and deletion) of the PARTROOT segment.

```
//NAME     JOB
//STEP1    EXEC   ????G
SYSTEM     SYSID=SAMP,SOMTX=OR
SEGMENT    ID=PA,LENGTH=50,NAME=PARTROOT,PARENT=0
FIELD      ID=KEY,KEY=YES,LENGTH=17,NAME=PARTNUMB
FIELD      ID=DESC,LENGTH=20,POS=27
GENERATE   OPT=NCLE,PGMID=OR
GENERATE   OPT=TPALL,TRXID=PA,DBPATH=PA,SPOS=SIMAGE,MODNAME=PARTROOT,
           IMAGE=PARTROOT
GENERATE   OPT=SGALL
```

Figure 11-10. Using Rules Generator Statements to Produce a Simple Nonconversational Transaction

The JCL procedure ????G executes the Rules Generator, where **????** is the installed ADFID (the default is MFC1; refer to the IMS Application Development Facility II Version 2 Release 2 Installation Guide). This procedure is supplied with IMSADF II.

The next sections describe the Rules Generator source statements shown in Figure 11-10 and give information you need to start using them. Refer to the IMS Application Development Facility II Version 2 Release 2 Application Development Reference for detailed descriptions of the various operands in each type of statement.

The SYSTEM Statement

```
SYSTEM    SYSID=SAMP,SOMTX=OR
```

The main operands of this statement are as follows:

SYSID Names the application system ID, the first four characters of the program load module for this application. **Required.**

The DL/I PSB, where applicable, and the IMS/VS transaction code, both have the same name as the program load module. The first two characters of the application system ID must be unique within the installation.

SOMTX Defines the last two characters of the program load module for this application (SAMPTOOR, in this case).

Note: SOMTX on the GENERATE statement (OPT=TPALL) overrides the operand on the SYSTEM statement.

The SEGMENT Statement

```
SEGMENT   ID=PA,LENGTH=50,NAME=PARTROOT,PARENT=0
```

The following operands are required for data base segments:

ID The two-character segment ID; must be unique within the application system.

LENGTH Segment length in bytes.

NAME Name of segment to be used in segment search arguments for DL/I calls. The same NAME value may be used for different segment definitions with different IDs. Such definitions are called aliases and are different views of the same data base segment.

PARENT The two-character ID of the parent segment in the data base. Root segments should have PARENT=0. DB2 tables and VSAM files also should have PARENT=0.

The FIELD Statement

```
FIELD     ID=KEY,KEY=YES,LENGTH=17,NAME=PARTNUMB
FIELD     ID=DESC,LENGTH=20,POS=27
```

All FIELD statements must have an ID and a LENGTH.

ID Field ID; two to four characters; must be unique within the segment.

KEY Indicates whether or not the field is a key field. The default is NO.

BYTES or LENGTH Length of stored field in bytes.

NAME Name of field to be used in segment search arguments for DL/I calls.

START or POSITION Position of field in the segment. The default is the byte immediately following the field defined in the preceding FIELD statement; if this is the first FIELD statement in the segment, the default is position 1.

KEY FIELDS: See "Key Fields" on page 2-6.

DECIMAL FIELDS: See "Decimal Fields" on page 2-7.

DATE FIELDS: See "Date Fields" on page 2-8.

The GENERATE Statement

```
GENERATE  OPT=NCLE,PGMID=OR
GENERATE  OPT=TPALL,TRXID=PA,DBPATH=PA,SPOS=SIMAGE,
          MODNAME=PARTROOT,IMAGE=PARTROOT
GENERATE  OPT=SGALL
```

This example includes three of the four types of GENERATE statement. The OPTIONS (OPT) operand determines which kind it is.

When setting up an application system, you need the following statement:

```
GENERATE  OPT=NCLE,PGMID=OR
```

The main operand of this statement is:

PGMID Defines the last two characters of the program load module for this application.

Note: SOMTX on the GENERATE statement (OPT=TPALL) overrides the operand on the SYSTEM statement.

The PGMID value must match the SOMTX operand value on the SYSTEM statement, or the GENERATE statement with OPT=TPALL must include a SOMTX value to override the operand on the SYSTEM statement, and there must be a separate GENERATE (with PGMID) statement for each different SOMTX operand value.

Transactions themselves are defined by the GENERATE statement with OPT=TPALL. The main operands of this statement are as follows:

TRXID Names the two-character transaction ID.

DBPATH Defines the target segments of the transaction. These are the segments for which the user will enter key information and which will be retrieved and updated according to the transaction mode selected by the user at the terminal.

SPOS=SIMAGE Indicates that a screen image definition is to be used.

MODNAME Sets the name of the display screen.

IMAGE Names the member of the screen image library (a PDS with DDNAME IMAGELIB) containing the screen image.

The Segment Layout and Segment Handler Rules will be generated when the Rules Generator encounters the following statement:

```
GENERATE  OPT=SGALL
```

It should be placed after all SEGMENT statements in the Rules Generator input.

PSEUDO SEGMENTS

It may be necessary to define working storage in a transaction for calculations or other data manipulation. Sometimes the fields in working storage will be displayed and updated by the user on the display screen. IMSADF II provides pseudo segments for this purpose. They are defined to the Rules Generator like data base segments, but without key fields, without parents, and without NAME operands. By default, the fields are displayed with MODE=5, but this can be changed just as for data base segments.

A pseudo segment is identified with a GENERATE statement (OPT=TPALL) using the operand TSEGS. For example:

```
SEGMENT   ID=CC,TYPE=PS
FIELD     ID=CLOR,TYPE=DEC,LENGTH=7
GENERATE  OPT=SGALL
GENERATE  TRXID=UV,DBPATH=IV,TSEGS=CC,OPT=TPALL,SPOS=SIMAGE,
          MODNAME=INVEN
```

PROGRAM FUNCTION KEYS

The support of program function (PF) keys in nonconversational processing is more extensive than that described in Chapter 6, "Complex Transactions." There is another option that allows IMS/VS commands (e.g., /FORMAT) to be entered when the user presses a PF key. (There is no analogous function for CICS BMS.)

To select this option, code PFKDATA=NO on the transaction GENERATE statement, do not use PFKNUMB, and code a PFKLIT operand for each PF key usage desired. Again, the Rules Generator will not check the validity of the PFKLIT value. (MFS rules must be followed.) For example:

Significant Rules Generator statement:

```
  GENERATE   TRXID=UV,DBPATH=IV,OPT=TPALL,
             SPOS=SIMAGE,MODNAME=INVEN,
             PFKLIT=(5='/FOR SAMP '),
             PFKLIT=(10='/FOR MFC1 ')
```

MEANING OF FIELD MODES

Nonconversational display screen formats are somewhat different from conversational screen formats. Only the information held on the screen will be saved from one transaction to the next. Therefore, IMSADF II must be able to read in keys and other required data from the display device whether these were saved from a previous display or just entered by the user. This is done for all fields by including the MODE=5 operand on the FIELD statement (or in the screen image definition). This makes the field both modifiable and premodified.

Note: "Premodified" means that data will be read in from the screen whether or not the user has amended it.

It is not required that all data be entered every time. Often, only keys and one or two fields are needed. By making all fields premodified (MODE=5), IMSADF II will treat them as changed, invoke the Auditor, and issue DL/I calls to update the corresponding data base segments even if these are not actually changed by the user. No DL/I calls to update data base segments are issued by IMSADF II when the transaction mode is 6 (Display); however, DL/I calls can still be made through the Auditor. The Auditor is not invoked after mapping the screen into the SPA when the transaction mode is 6 unless a field with MODE=4 is specified in the transaction.

A field with MODE=4 will not be premodified. Data in field of MODE=4 will be read in only if the user enters data into it. In addition, for compatibility with conversational processing, a field of MODE=4 will be modifiable even in transaction mode 6. If the user specifies a MODE=4 field on the screen, the Auditor will be called. Using the Auditor in this way in transaction mode 6 can be useful for transaction switching or for unusual processing requirements.

If the transaction mode is not 6, all MODE=5 fields and modified MODE=4 fields will be mapped from the screen into the SPA. The Auditor will be invoked and IMSADF II will issue DL/I calls to update data base segments.

For transaction mode 6, only pseudo segment fields that are MODE=5 or modified MODE=4 will be mapped from the screen to the SPA. No displayable DBPATH fields will be mapped, whatever their field mode. The Auditor will be invoked if a MODE=4 field was specified. No DL/I calls to update data base segments will be issued by IMSADF II.

SUMMARY OF SYNTAX CONVENTIONS

- Start in any column (1-71) and use columns 1 to 71.
- Leave a space between control statement keywords and operands.
- Separate comments from statements by one or more blank lines. An asterisk in column 1 marks a comment line.
- Mark continuations with a comma/blank combination. The next line can start in any column (1-71).
- Do not continue multi-valued operands (using parentheses) over two lines. Instead, close the parentheses and repeat the operand on the next line: OPT=(INTR,SEGD),OPT=KEYD is the same as OPT=(INTR,SEGD,KEYD).

DYNAMIC RULES

When IMSADF II is installed, three data bases are set up that contain dynamic rules. These data bases are maintained online or in batch using an application system provided with the product. They are referenced during the execution of application systems developed using IMSADF II.

The three data bases are shown in Figure 11-11.

Data Base	Description
Sign-on Profile	For nonconversational applications, this data base is used only to collect information messages for batch printing.
Audit	Controls validation of data field format and content; allows specification of calculations, logic operations, and data manipulation; provides for additional security checking by key range or field values; and supports table definition.
Message	Used mainly in conjunction with rules in the Audit Data Base. Contains text and format of error messages and information.

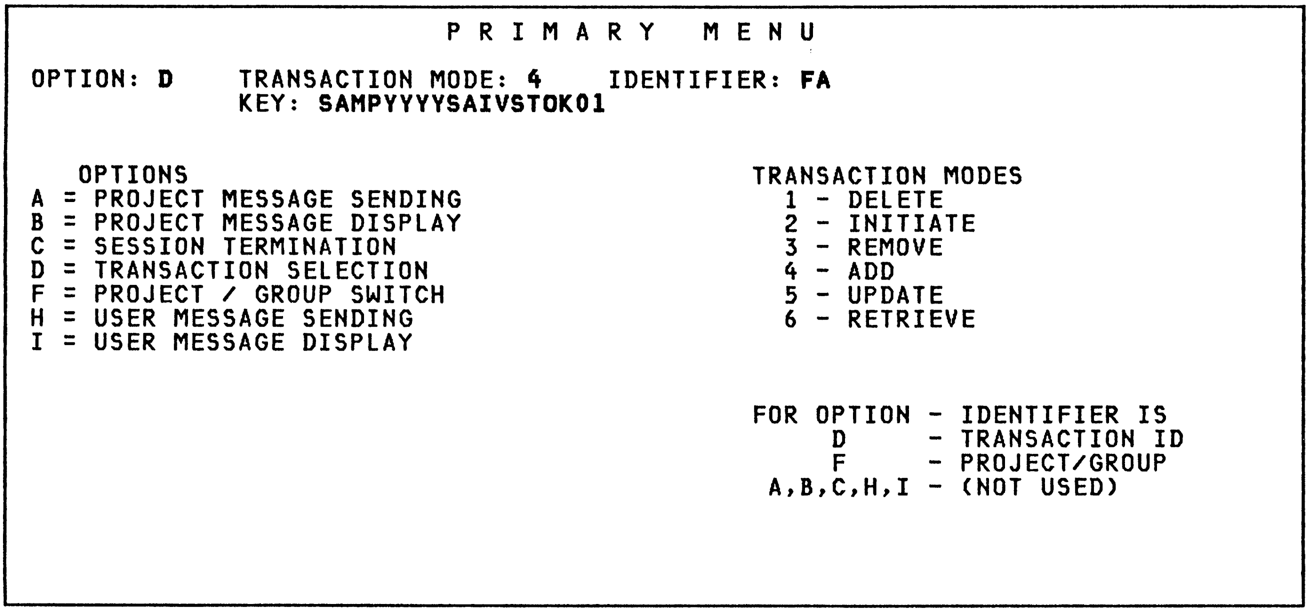
Figure 11-11. The Dynamic Rules Data Bases

Most functions needing dynamic rules require information from both the Audit and Message Data Bases. Applications can be implemented, however, without using dynamic rules at all.

Entering rules into these data bases is part of application development and maintenance and will therefore normally be done by the same person that writes the static rules. Dynamic rules can be entered through batch utilities that accept statements in a high level audit language and compile them into a form in which they can be loaded onto the data bases.

An application system is provided with IMSADF II for the purpose of creating and maintaining dynamic rules online through IMS/VS. This system treats the IMSADF II dynamic rules data bases as if they were application data bases. It is implemented using conversational processing and is hence used somewhat differently from the previous example.

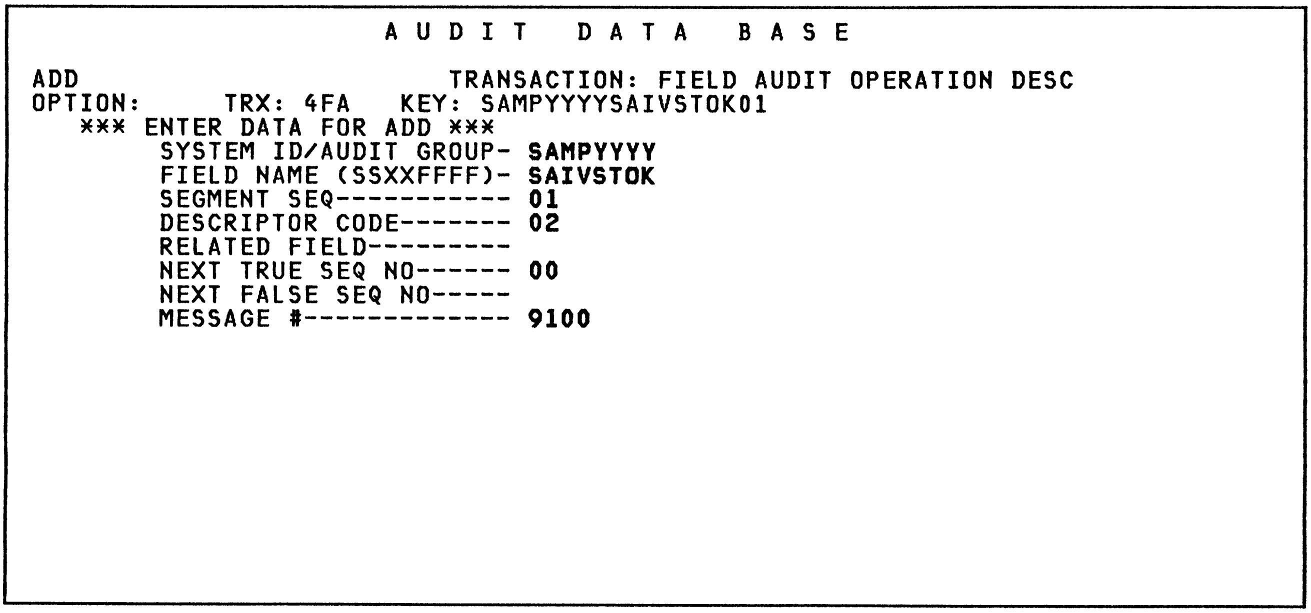
Suppose you want to create a segment in the Audit Data Base having a key of SAMPYYYYSAIVSTOK01. Figure 11-12 shows how to request this. Only OPTION D (Transaction Selection) and OPTION C (Session Termination) need be used to update the data bases for nonconversational transactions. The transaction modes are the same as in conversational transactions. Transaction ID FA is one of many that exist in this application system. If a transaction ID is not entered, a menu describing all valid transaction IDs will be shown to enable you to make a selection. Data base browsing capabilities, as described in Chapter 1, "IMSADF II Concepts and Overview," can also be used.



```
                        P R I M A R Y   M E N U

OPTION: D    TRANSACTION MODE: 4    IDENTIFIER: FA
             KEY: SAMPYYYYSAIVSTOK01


   OPTIONS                                       TRANSACTION MODES
A = PROJECT MESSAGE SENDING                        1 - DELETE
B = PROJECT MESSAGE DISPLAY                        2 - INITIATE
C = SESSION TERMINATION                            3 - REMOVE
D = TRANSACTION SELECTION                          4 - ADD
F = PROJECT / GROUP SWITCH                         5 - UPDATE
H = USER MESSAGE SENDING                           6 - RETRIEVE
I = USER MESSAGE DISPLAY


                                              FOR OPTION - IDENTIFIER IS
                                                   D     - TRANSACTION ID
                                                   F     - PROJECT/GROUP
                                               A,B,C,H,I - (NOT USED)
```

Figure 11-12. Using the Primary Option Menu to Select Transaction FA

When you press ENTER, a Data Display screen will appear (see Figure 11-13). Data entered here will go onto the data base.



```
                      A U D I T   D A T A   B A S E

ADD                        TRANSACTION: FIELD AUDIT OPERATION DESC
OPTION:     TRX: 4FA   KEY: SAMPYYYYSAIVSTOK01
   *** ENTER DATA FOR ADD ***
        SYSTEM ID/AUDIT GROUP- SAMPYYYY
        FIELD NAME (SSXXFFFF)- SAIVSTOK
        SEGMENT SEQ----------- 01
        DESCRIPTOR CODE------- 02
        RELATED FIELD---------
        NEXT TRUE SEQ NO------ 00
        NEXT FALSE SEQ NO-----
        MESSAGE #------------- 9100
```

Figure 11-13. Adding a Segment to the Audit Data Base

To return to the Primary Option Menu, enter OPTION C on the screen. On the Primary Option Menu, OPTION C will terminate the session.

The entries in Figure 11-13 are explained in Chapter 4, "The Auditor and the Audit Data Base." The next section will introduce that chapter.

AUDITING FIELDS

Data validation, calculations, and other processing against fields are performed by the Auditor, which is a part of the transaction driver. The operations it performs are controlled by rules stored in the Audit Data Base. In addition, there are certain operands to be coded on Rules Generator statements to request that audit operations be performed. If no such operands are coded, the Auditor will simply validate the data entered by the user according to the data type coded on the Rules Generator FIELD statements. If errors are found, the fields in error are highlighted and the user is invited to select the next logical page to see the error messages, as described in Chapter 1, "IMSADF II Concepts and Overview."

The Auditor may be called both during and after key selection, before the data display is shown to the user. The phase of auditing that takes place during key selection is known as **key audit**.

You can use this phase to edit keys, to cause a switch to another transaction based on the value of the key that the user has entered, and to validate keys and impose security by key range and user ID or terminal ID. If errors are detected during key audit, the keys in error are highlighted, and the user is invited to select the next logical page to display the error messages.

In conversational processing, the Auditor can also control secondary key selection (data base browsing). That function is not available in nonconversational processing.

The next phase of auditing is termed **preaudit**. This takes place after the DBPATH segments have been retrieved through key selection. Here, it may be necessary to prevent some users from updating individual fields, to convert certain data fields to a different format for viewing, to initialize fields in a nonstandard way, or to perform data base retrievals for some or all of the segments. Again, error messages are on the next logical page.

The Auditor is called in transaction modes 1 to 5 after preaudit, before the transaction driver issues DL/I calls to update the data bases. If errors are found, the data base is not updated. When the user has entered corrections, the Auditor is called again and all audit rules are performed once more. Several iterations can take place before the data bases are finally updated.

After the updates have been made successfully, it is possible for the user to enter further amendments. The Auditor then validates and processes them and further data base updates can be performed.

If one or more fields of MODE=4 have been defined (so that they are not premodified), the user can enter data in them, even in transaction mode 6. The presence of a MODE=4 field will cause the Auditor to be called in the PROCESS phase, as well as in key audit and preaudit.

Read Chapter 4, "The Auditor and the Audit Data Base" and Chapter 5, "Message Sending and Display" for more information.

COMPLEX TRANSACTIONS

This chapter has already described transactions that display and update multiple hierarchical paths in multiple data bases. This section presents more advanced application functions and deals with arbitrary combinations of inserting, deleting, and replacing segments.

Issuing DL/I calls during audit operations gives control over complex updating as well as providing an important capability in data validation and a method of processing twins (multiple segment occurrences).

MULTI-PATH TRANSACTIONS

As you know, standard processing transactions are defined via the GENERATE statement of the Rules Generator. For example:

```
GENERATE  TRXID=PI,OPT=TPALL,DBPATH=(CY,OR),
          TSEGS=(WO,PD),SPOS=SIMAGE,IMAGE=SAMPPI
```

The segments named in the DBPATH operand are defined in preceding SEGMENT statements and are the target segments of the transaction. The data base layouts assumed are shown in Figure 11-14. WO is the ID of a pseudo segment.

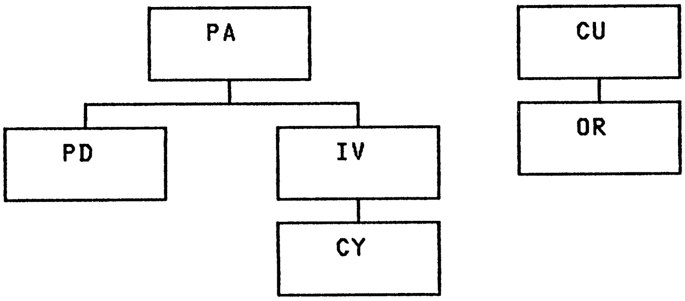


Figure 11-14. Data Bases Used in Examples

If any field from the CU segment is displayed (by inclusion in the screen image), the CU segment will be included in the transaction and will be updated if data is changed by the online user or by audit rules. The same applies to the PA and IV segments. If no field from a segment (e.g., IV) is displayed, but audit logic will be required to access or update it, include the segment in the DBPATH thus:

```
DBPATH=(CY,IV,OR)
```

The target segments are still CY and OR as they are the lowest in each hierarchical path. Collectively, the segments named or implied by the DBPATH operand are called DBPATH segments. The user must enter their keys when using the transaction and together their keys constitute the concatenated key of the transaction.

Segments named in TSEGS are either pseudo segments or data base segments to be retrieved by the Auditor under control of audit rules or by a special processing program. The keys of data base TSEGS are mapped from the screen input prior to any required primary key audits and/or preaudits.

Updating of DBPATH segments is controlled by the transaction mode selected by the end user. (1 and 2 are interchangeable with 3 and 4, respectively.)

Mode 5: The user must enter keys of existing DBPATH segments and the segments are displayed. If he changes data, the changed segments will be updated on the data base. If auditing changes data, those segments will also be updated.

Mode 4: The user must enter keys of existing nontarget DBPATH segments and the key of at least one target segment that does not exist on the data base so that it can be inserted. For the other target segments, he can enter an existing key or one that does not exist. If the user changes data, changed segments are replaced and new segments are inserted. Again, auditing changes also lead to segments being updated (replaced or inserted).

Mode 3: The user must enter keys of existing DBPATH segments. For transactions with a single target segment, the Auditor will be called and the segment will be deleted regardless of whether or not the user changes data on the screen. If the user or the Auditor changes data in other segments, they will be replaced.

For transactions with multiple target segments, mode 3 is just like mode 5, except that in mode 3 the Auditor will be called (and can therefore cause updates) whether or not the user changes data.

DELETE ELIGIBILITY

To define a transaction that deletes segments other than the target segment in a single path transaction, you must:

- Define delete eligibility against those segments
- code DL/I calls through the audit operation to delete the segments

Use the DLET operand on the GENERATE statement for the transaction.

In the following example the audit rule checks for a transaction mode of 3 before deleting. The audited field is a nondisplayed dummy field in the pseudo segment WO. The user receives the display with the message **PRESS ENTER TO DELETE DATE**. When he does so, the CY and OR segments are deleted.

Significant Rules Generator statements:

```
SEGMENT   ID=WO,TYPE=PS,LENGTH=1,DISP=NO
FIELD     ID=DUMY,LENGTH=1,AFA=YES
GENERATE  OPT=SGALL
GENERATE  TRXID=OM,OPT=TPALL,DBPATH=(PD,CY,OR),MODNAME=ORDMAINT,
          TSEGS=WO,DLET=(CY,OR),SPOS=SIMAGE,IMAGE=ORDMAINT
```

High level audit language:

```
SYSID = SAMP
SEGID = WO
FIELD = DUMY
IF MODE = 3
  IF DLET KEYFIELD CY OK
  NOP
  ENDIF
  IF DLET KEYFIELD OR OK
  NOP
  ENDIF
ENDIF
```

Generated operation descriptors:

```
Audit root key:          SAMPYYYYSAWODUMY
Audit operation desc:    0167        0200        TRANSACTION MODE=3
Audit data desc.:          0001(3)
Audit operation desc:    0236KEYFIELD03  1025    DELETE CY SEGMENT
Audit data desc:           0001(CY,DLET)
Audit operation desc:    0336KEYFIELD00  1025    DELETE OR SEGMENT
Audit data desc:           0001(OR,DLET)
```

Note: The related field KEYFIELD is a special value recognized by the DL/I call audit operation as meaning the key of the segment already retrieved.

If an attempt is made to delete a segment using the audit operation and the segment is not eligible for deletion, the deletion is not done, the audit operation returns false and a DL/I status code of AM is set.

The deletion is not performed immediately; the operation merely sets a flag which is later acted on by the transaction driver when performing any other data base updates.

Segments named in the DLET operand must be DBPATH or TSEG segments.

INSERT ELIGIBILITY

Insert eligibility has nothing to do with DL/I insert calls from audit operations, which can be coded regardless of insert eligibility. Insert eligibility alters data base updating to allow insertions of DBPATH segments (target or not) in any transaction mode except 6. Segments are made eligible through the ISRT operand of the transaction GENERATE statement.

Mode 5: The user must enter keys of DBPATH segments, but if a segment is eligible for insertion, he is free to enter a key that does not exist on the data base. If he does and proceeds to enter data into it, the segment will be inserted. For an existing key, the segment will be replaced.

Mode 4: All modifiable segments can be inserted regardless of insert eligibility.

Mode 3: As in mode 5, the eligible segments can be inserted or replaced depending on the keys entered by the user.

DL/I CALLS FROM THE AUDITOR

See "DL/I Calls from the Auditor" on page 6-16. The examples given there apply to nonconversational processing, except that the transaction GENERATE statements:

- Must be coded OPT=TPALL instead of OPT=CVALL
- Must be accompanied by screen image definitions (SPOS=SIMAGE)

TRANSACTION SWITCHING

See "Transaction Switching" on page 6-9.

In nonconversational processing, the means for passing keys and data from one transaction to the next is different. The SPAKEYID keyword is not used; neither is it necessary to write an exit routine to pass data. However, an exit is still required if the new TRXID must be set from a pseudo segment field rather than a literal.

Keys and data are passed by virtue of a naming convention. IMSADF II compares the field names (segment ID plus field ID) in the old transaction with those in the new by comparing the relevant rules. For those names that match, the values are mapped across with any necessary conversions. This applies to keys and to data.

The display screen of the new transaction is then presented to the user for completion; when the user presses ENTER, the new transaction is triggered.

SECONDARY TRANSACTIONS AND IMS/VS MESSAGE ROUTING

The description in Chapter 7, "Secondary Transactions and IMS/VS Message Routing" applies to nonconversational processing.

In addition, a message can be sent back to the input logical terminal immediately by coding the special destination IOPCB in the routing information placed in the Message Data Base. This is required only for nonresponse-type transactions.

NONRESPONSE TRANSACTIONS

A different kind of nonconversational transaction can be implemented with IMSADF II. This is the kind that does not send a response back to the originating logical terminal or, at least, does not send a response back in the same format as the input.

Under &ims, any output message from such a transaction must be generated using the secondary transaction facility described in Chapter 7, "Secondary Transactions and IMS/VS Message Routing." Messages, such as Auditor error messages, are sent to a printer logical terminal.

Under CICS/OS/VS, any output message from such a transaction is sent to the device associated with the Transient Data queue that triggered the transaction.

Nonresponse transactions can be entered directly from a terminal but is more suitable to use in transactions triggered from other transactions.

TRANSACTION FORMAT

Each transaction begins with a 12-character transaction mode and identifier of the form:

sssTOcc mtx

where:

ssss is the application system ID
TO is a literal (in a BMP use BO)
cc is the cluster code
m is the transaction mode 1 to 6 (preceded by a blank space)
tx is the transaction ID

Chapter 7, "Secondary Transactions and IMS/VS Message Routing" gives examples of rules to trigger a secondary transaction. An example of the actual transaction message would be:

```
SAMPTOSC 5SC02AN9724686370241023
```

RULES

As for all other types of processing, the Rules Generator expects SYSTEM, SEGMENT, FIELD, and GENERATE statements. The SYSTEM and SEGMENT statements are the same as for response transactions. The FIELD statements require the following additional operands:

FLDPOS Starting position of the field in any nonresponse (or batch) transaction in which the field is used. First available position is 13.

ILENGTH The input length of the field, if this differs from the LENGTH operand value.

ITYPE The input data type. The default is alphanumeric. Allowed values are the same as those for the TYPE operand. Data conversions will be performed.

These operands will be ignored when rules for response transactions are being generated; therefore, the same definitions can be used to create both kinds.

As for all types of processing, GENERATE statements for segments and transactions are required. Those for segments are the same and need not be repeated. Those for transactions differ from response transactions in that no screen image is defined and no MFS statements are produced by the Rules Generator. Therefore, no SPOS operand is coded and OPT=TPIT (Teleprocessing Input Transaction Rule) must be requested.

Example

Suppose that the SC transaction needs to retrieve the PA and PD segments in the sample data base. The keys of these segments are in the transaction. The data field is to be read into a pseudo segment. The Rules Generator statements are as follows:

```
  SYSTEM SYSID=SAMP,SOMTX=OR,ASMLIST=NOGEN
  SEGMENT PARENT=0,ID=PA,NAME=PARTROOT,LENGTH=50
    FIELD ID=KEY,LENGTH=17,KEY=YES,NAME=PARTKEY,FLDPOS=13
    FIELD ID=DESC,LENGTH=20,POS=27,FLDPOS=26
  SEGMENT ID=PD,NAME=STANINFO,LENGT=85,PARENT=PA
    FIELD ID=KEY,LENGTH=2,KEY=YES,NAME=STANKEY,FLDPOS=30
  SEGMENT ID=WW,TYPE=PS
    FIELD ID=IW,LENGTH=1,FLDPOS=32
  GENERATE SEG=(PA,PD),OPT=(SEGH,SEGL)
  GENERATE SEG=WWC,OPT=SEGL
  GENERATE TRXID=SC,DBPATH=PD,OPT=TPIT,TSEG=WW
```

SPECIAL PROCESSING

Processing controlled solely by rules is known as "standard processing." Special processing is provided as an extension to standard processing. Special processing routines (SPRs) are written in COBOL, PL/I, or Assembler. They are executed under the control of the transaction driver, but perform functions that the transaction driver cannot. Rules are coded to control the transaction driver in much the same way for special as for standard processing.

The principal use of special processing is for complex, application dependent logic for which audit rules are too cumbersome. For example, the Auditor does not provide array manipulation. Coding that applies to repeated fields or segments must be coded repetitively for each occurrence with different names.

The special processing routine is invoked after the DBPATH segments have been retrieved and the data has been moved in from the screen. Figure 11-15 shows the overall flow.

On the GENERATE statement for the transaction, code the following additional operands:

SPECIAL=YES   Requests special processing.

LANGUAGE=COBOL   Programming language in which the SPR is written.
         PL/I   COBOL is the default; ASMINT means Assembler.
         ASMINT

The description of special processing in Chapter 8, "Special Processing," beginning at "Program Calls" on page 8-3, is applicable to nonconversational processing.

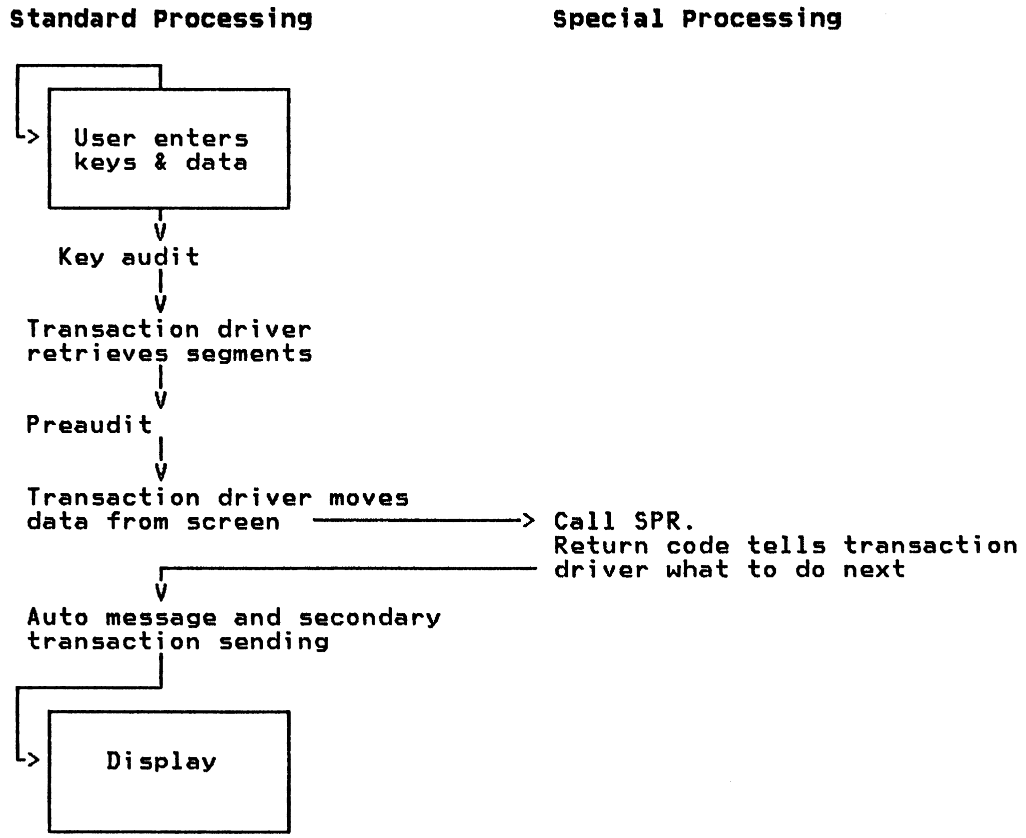

Standard Processing
Special Processing
User enters keys & data
Key audit
Transaction driver retrieves segments
Preaudit
Transaction driver moves data from screen
Call SPR.
Return code tells transaction driver what to do next
Auto message and secondary transaction sending
Display

Figure 11-15. Special Processing Flow (Nonconversational Processing)

PROGRAM LINKAGE

"Program Linkage" on page 8-21 applies to nonconversational processing except that on GENERATE statements that request a link edit, OPT=TPLE must be coded instead of OPT=SPLE.

RETURN CODES

The return codes passed by the SPR to the transaction driver also differ. The SPR sets the return code to tell the transaction driver what to do next. The return code must be set through the usual Operating System method (COBOL: RETURN-CODE, PL/I: PLIRETC, Assembler: Register 15), not in SPARTNCD.

Code Transaction Driver Action

1 Generate secondary transactions as required according to the setting of SPASECTX. Control is immediately returned to the special processing routine.

3 Print (to printer LTERM) the message in SPAERMSG if this is a nonresponse Input Transaction Rule. Display the message in SPAERMSG according to Output Format Rule specifications if this is a response Input Transaction Rule. Read the next message.

4 Print (to printer LTERM) SPECIAL PROCESSING SUCCESSFULLY COMPLETED if this is a nonresponse Input Transaction Rule. Display the message if this is a response Input Transaction Rule according to Output Format Rule specifications. Read the next message.

5 The transaction ID named in SPACGTRX is set up as the next transaction ID to be processed. The display screen associated with the ID in SPACGTRX will be displayed and will contain only the fields defined in the Input Transaction Rule of the current transaction.

8 Print (to printer LTERM) or display audit error message and read next message.

12 SPR has generated a message. Read next message.

24 Issue ROLL call to back out any updates. Pseudo ABEND caused.

28 Generate and print (to printer LTERM) or display the error message returned from the segment handler and read next transaction.

32 Read the next message.

Note: Secondary transaction generation is invoked for any of the above return codes if SPASECTX is set to a nonzero value.

IMS/VS CONSIDERATIONS

Each special processing transaction ID must have an associated IMS/VS transaction code with name:

ssssT0tx

where:

ssss is the application system ID
T0 is a literal
tx is the transaction ID

The Rules Generator GENERATE OPT=TPLE statement will produce an executable load module with the same name (ssssT0tx). A PSB (or DB2 plan) with the same name must also be provided. Apart from the difference in name, the coding of the PSB and IMS/VS system definition macros is the same as for standard processing. For more information consult the IMS Application Development Facility II Version 2 Release 2 Application Development Reference.

CHAPTER 12. HELP FACILITY

A HELP facility is available to the user of IMSADF II conversational processing. Under this facility the user may define and invoke HELP text relating to either the current processing screen or error and warning message(s). The HELP text is stored in the Message Data Base and is available, upon operator request, to the following screens:

- Sign-On
- Primary Option Menu
- Secondary Option Menu
- Primary and Secondary Key Selection
- Segment Display
- Text Utility
- Error Display (Resulting from Audit/Deformatting Errors)

SCREEN HELP FACILITY

Each of the above screens (except the Error Display) may have a tailored description of function and input requirements displayed upon the entry of a '?'.

- In the USERID, PROJECT, or GROUP fields of the Sign-On screen,
- In the OPTION field of the Primary Option Menu, Primary Key Selection, Secondary Key Selection, Segment Display, Text Utility and
- In the SELECT field of the Secondary Option Menu screen.

The HE and HT segments are used to store Screen HELP text. The HT segment may contain up to 20 lines of text plus a header.

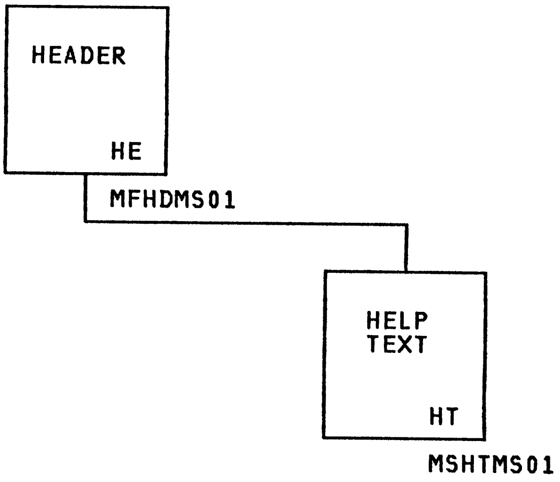
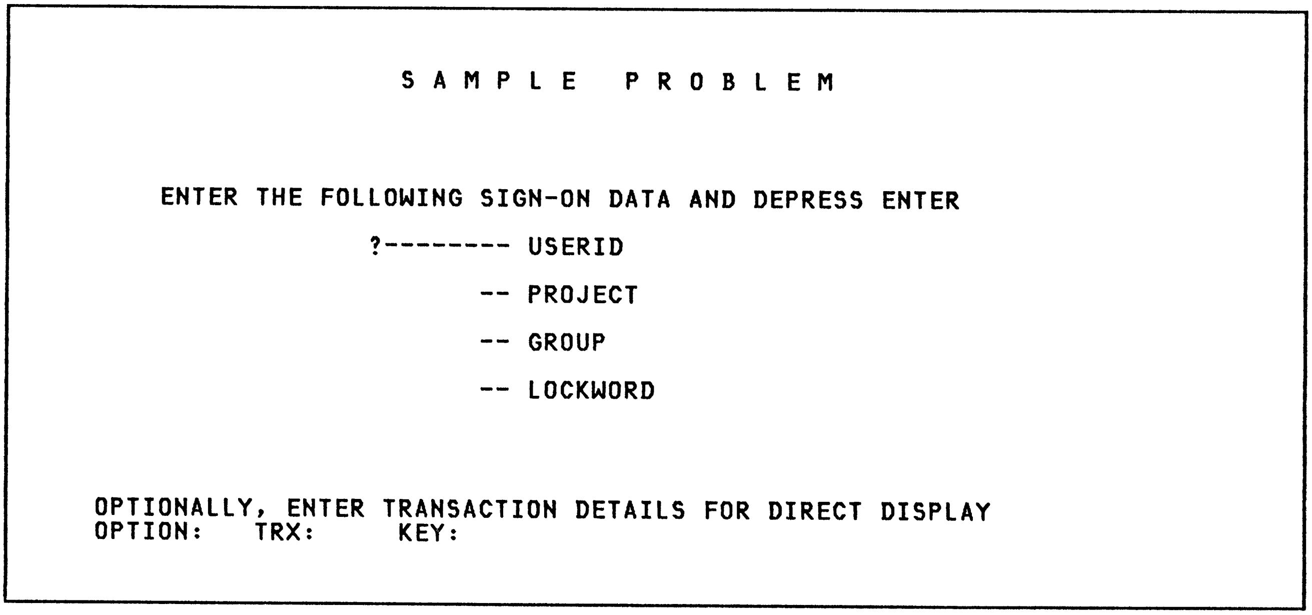


Figure 12-1. Screen HELP Segments in Message Data Base

Each system has its own set of tailored HELP screens. Each transaction may have HELP text describing the Segment Display screen and HELP text describing the Primary/Secondary Key selection screens.

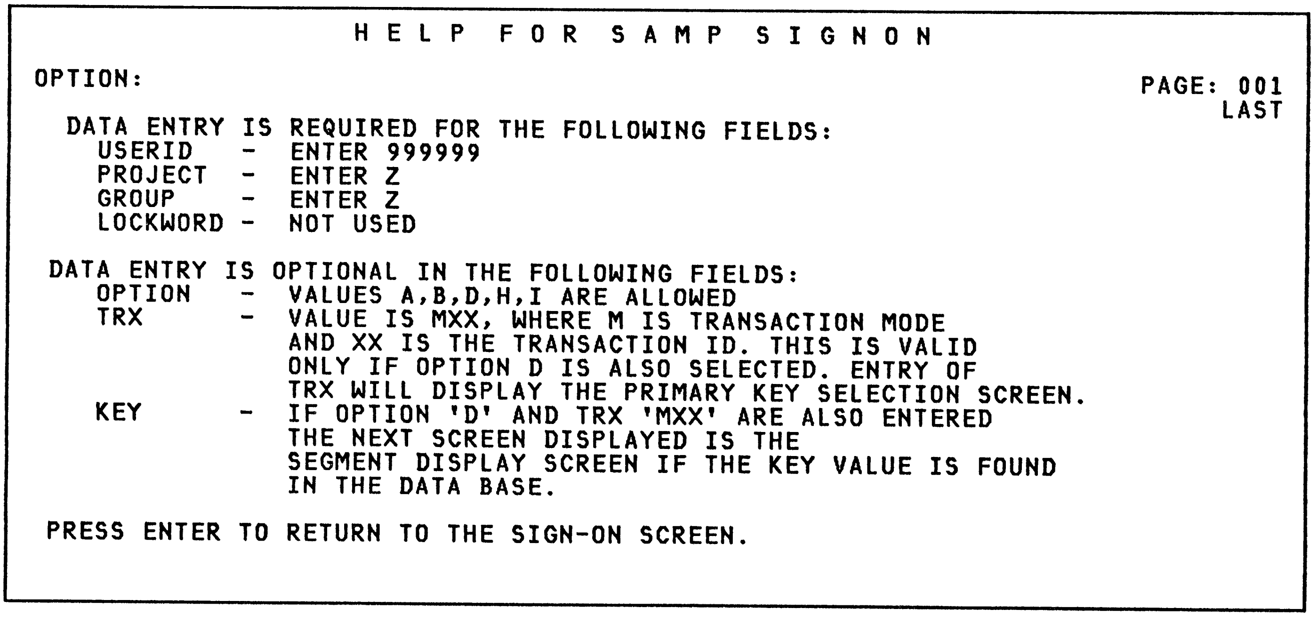
When a '?' is entered on a screen,



```
                         S A M P L E   P R O B L E M


          ENTER THE FOLLOWING SIGN-ON DATA AND DEPRESS ENTER
                       ?-------- USERID

                              -- PROJECT

                              -- GROUP

                              -- LOCKWORD



      OPTIONALLY, ENTER TRANSACTION DETAILS FOR DIRECT DISPLAY
      OPTION:   TRX:     KEY:
```

Figure 12-2. HELP request on SAMPLE Sign-On Screen

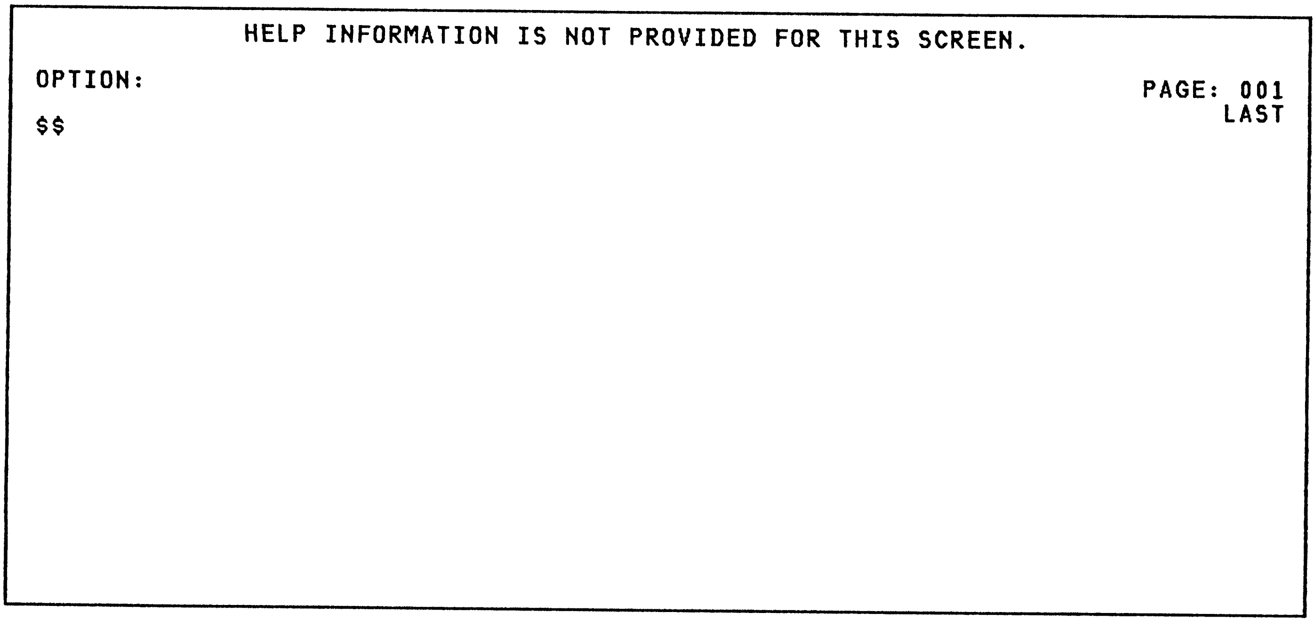
either the HELP screen is displayed:



```
                       H E L P   F O R   S A M P   S I G N O N
 OPTION:                                                                PAGE: 001
                                                                             LAST
   DATA ENTRY IS REQUIRED FOR THE FOLLOWING FIELDS:
     USERID   -  ENTER 999999
     PROJECT  -  ENTER Z
     GROUP    -  ENTER Z
     LOCKWORD -  NOT USED

  DATA ENTRY IS OPTIONAL IN THE FOLLOWING FIELDS:
     OPTION   -  VALUES A,B,D,H,I ARE ALLOWED
     TRX      -  VALUE IS MXX, WHERE M IS TRANSACTION MODE
                 AND XX IS THE TRANSACTION ID. THIS IS VALID
                 ONLY IF OPTION D IS ALSO SELECTED. ENTRY OF
                 TRX WILL DISPLAY THE PRIMARY KEY SELECTION SCREEN.
     KEY      -  IF OPTION 'D' AND TRX 'MXX' ARE ALSO ENTERED
                 THE NEXT SCREEN DISPLAYED IS THE
                 SEGMENT DISPLAY SCREEN IF THE KEY VALUE IS FOUND
                 IN THE DATA BASE.

  PRESS ENTER TO RETURN TO THE SIGN-ON SCREEN.
```

Figure 12-3. HELP for SAMPLE Sign-On Screen

or, if HELP text is not provided, an informational message is displayed:

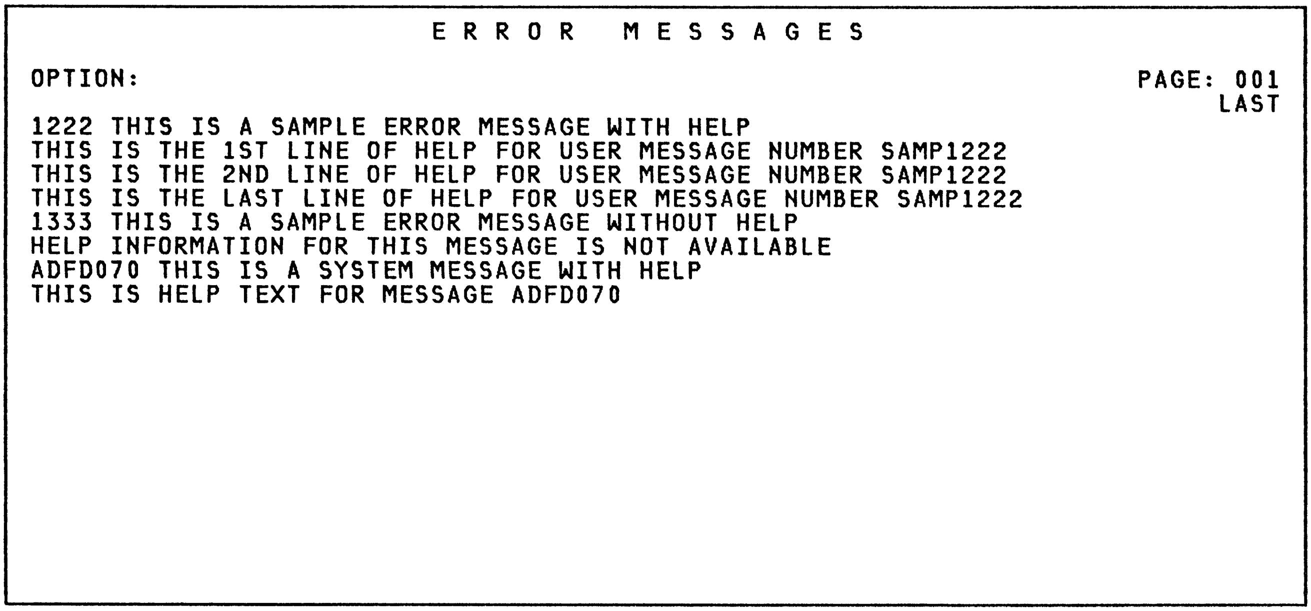


```
               HELP INFORMATION IS NOT PROVIDED FOR THIS SCREEN.
OPTION:
                                                                    PAGE: 001
$$                                                                       LAST
```

Figure 12-4. Sample of HELP not provided

MESSAGE HELP FACILITY

Error and warning messages in the Message Data Base may have HELP text associated with them. HELP is invoked through the entry of a '?' in the Option field of either the Error screen or the screen creating the error. The Error screen is displayed showing all error or warning messages followed by their corresponding HELP text.



```
                              E R R O R   M E S S A G E S

OPTION:                                                                  PAGE: 001
                                                                              LAST
1222 THIS IS A SAMPLE ERROR MESSAGE WITH HELP
THIS IS THE 1ST LINE OF HELP FOR USER MESSAGE NUMBER SAMP1222
THIS IS THE 2ND LINE OF HELP FOR USER MESSAGE NUMBER SAMP1222
THIS IS THE LAST LINE OF HELP FOR USER MESSAGE NUMBER SAMP1222
1333 THIS IS A SAMPLE ERROR MESSAGE WITHOUT HELP
HELP INFORMATION FOR THIS MESSAGE IS NOT AVAILABLE
ADFD070 THIS IS A SYSTEM MESSAGE WITH HELP
THIS IS HELP TEXT FOR MESSAGE ADFD070
```

Figure 12-5. Sample of error messages with HELP

Message HELP text is maintained as one or more child segments (MH) of the last error or warning message text segment(s) (SY).

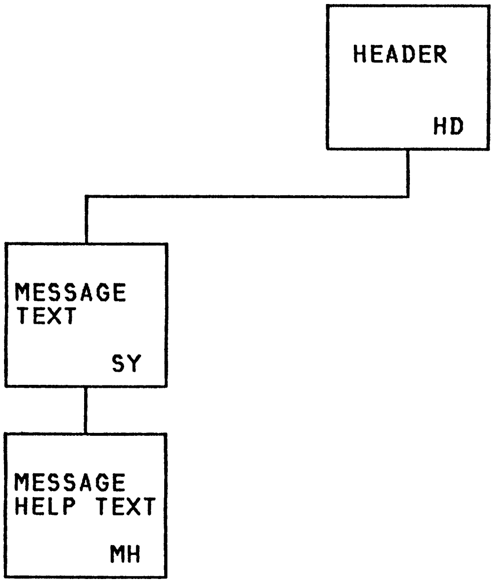


Figure 12-6. Message Data Base

CREATING HELP TEXT WITH ONLINE TRANSACTIONS

Three conversational transactions are provided to allow the creation of screen and message HELP text.

Transaction 'HE' creates the header segment for the screen HELP text.

HELP Header (HE) (Alias of HD - Segment Length = 78 bytes)

Position	Length	Data Type	Data Description
1	8	alphanum	Segment sequence field (key)
9	70		reserved.

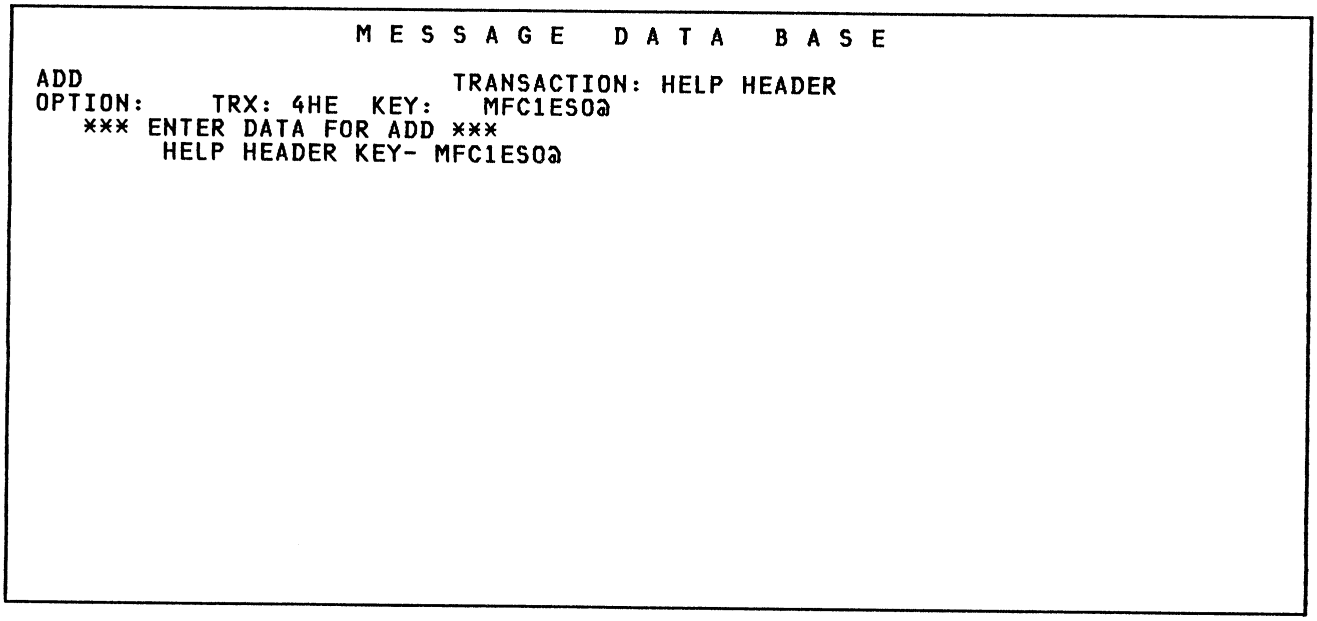
The key of the HE segment is formatted according to the screen for which the HELP text is created.

KEY FORMAT.	ASSOCIATED SCREEN
ssssESO@ or ????ESO@	Signon Screen. The ????ESO@ format is used when a general signon screen is used and the Application SYSTEM ID is not yet input.
ssssEOM@	Primary Option Menu
ssssESM@	Secondary Option Menu
ssssPxx@	Primary and Secondary Key Selection
ssssHxx@	Segment Display and Text Utility

where:

ssss is the application system id
???? is the installed ADFID
xx is the currently processed transaction id
E,P,H,@ are constants to indicate the HELP function

The 'HE' transaction screen allows for entry of a formatted key as follows.



```
                         M E S S A G E   D A T A   B A S E
ADD                                 TRANSACTION: HELP HEADER
OPTION:    TRX: 4HE   KEY:   MFC1ESO@
   *** ENTER DATA FOR ADD ***
        HELP HEADER KEY- MFC1ESO@
```

Figure 12-7. HE-HELP HEADER Generation Screen

Transaction 'HT' creates the HELP text for the screen HELP facility.

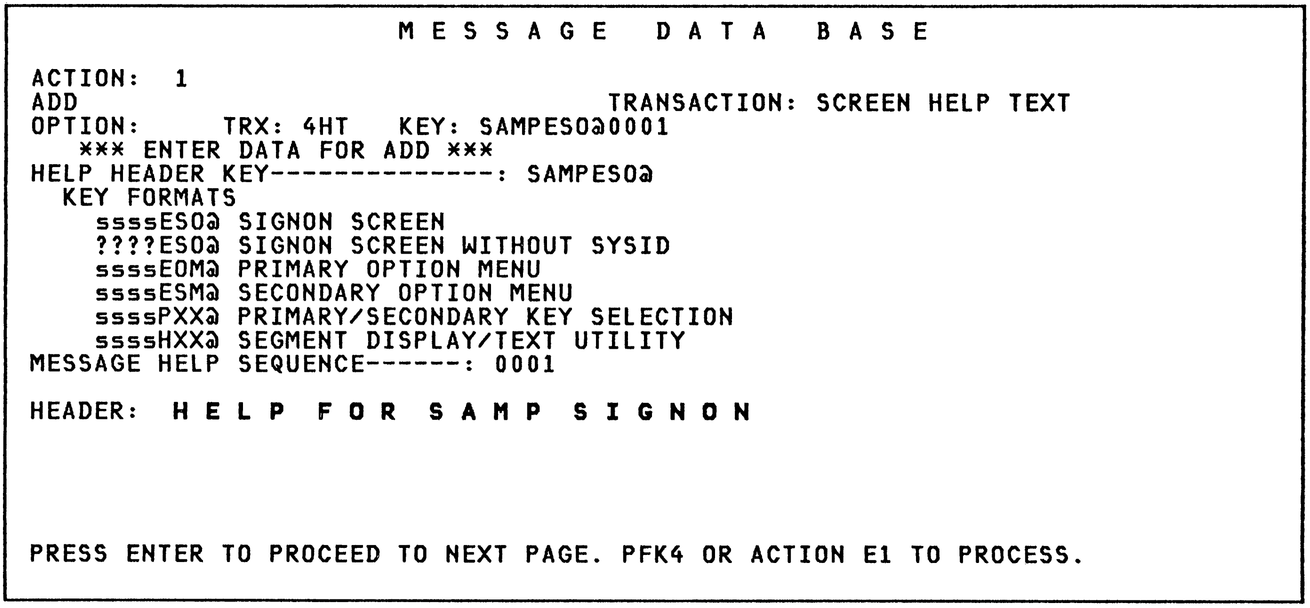
Screen HELP Text (HT) Layout (Segment Length = 1644 bytes)

Position	Length	Data Type	Data Description
1	4	alphanum	segment sequence field (key)
5	60	alphanum	header for line 1 of each page.
The following represents 20 lines on the HELP screen.			
65	79	alphanum	text that represents line 5
144	79	alphanum	text that represents line 6
223	79	alphanum	text that represents line 7
302	79	alphanum	text that represents line 8
381	79	alphanum	text that represents line 9
460	79	alphanum	text that represents line 10
539	79	alphanum	text that represents line 11
618	79	alphanum	text that represents line 12
697	79	alphanum	text that represents line 13
776	79	alphanum	text that represents line 14
855	79	alphanum	text that represents line 15
934	79	alphanum	text that represents line 16
1013	79	alphanum	text that represents line 17
1092	79	alphanum	text that represents line 18
1171	79	alphanum	text that represents line 19
1250	79	alphanum	text that represents line 20
1329	79	alphanum	text that represents line 21
1408	79	alphanum	text that represents line 22
1487	79	alphanum	text that represents line 23
1566	79	alphanum	text that represents line 24

The 'HT' transaction screen allows for entry of up to 20 lines of HELP text plus a screen header. Multiple HT segments may be added.

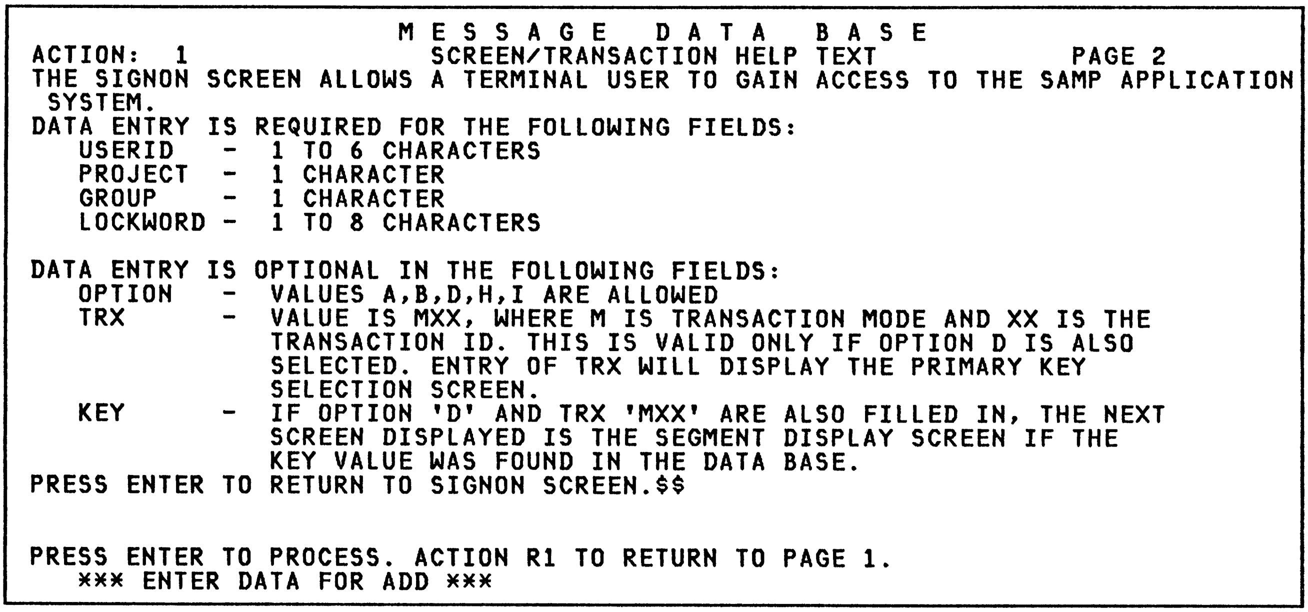
Note that the HEADER need only be specified in the first HT segment.

The HT screen in Figure 12-8 demonstrates input to build HELP text for the Sign-On screen for application system SAMP.



```
                          M E S S A G E   D A T A   B A S E

ACTION:  1
ADD                                   TRANSACTION: SCREEN HELP TEXT
OPTION:     TRX: 4HT   KEY: SAMPESO@0001
   *** ENTER DATA FOR ADD ***
HELP HEADER KEY--------------: SAMPESO@
  KEY FORMATS
    ssssESO@ SIGNON SCREEN
    ????ESO@ SIGNON SCREEN WITHOUT SYSID
    ssssEOM@ PRIMARY OPTION MENU
    ssssESM@ SECONDARY OPTION MENU
    ssssPXX@ PRIMARY/SECONDARY KEY SELECTION
    ssssHXX@ SEGMENT DISPLAY/TEXT UTILITY
MESSAGE HELP SEQUENCE------: 0001

HEADER:  H E L P  F O R  S A M P  S I G N O N




PRESS ENTER TO PROCEED TO NEXT PAGE. PFK4 OR ACTION E1 TO PROCESS.
```

Figure 12-8. HT-HELP Generation Screen (Page 1)



```
                        M E S S A G E   D A T A   B A S E
ACTION: 1                  SCREEN/TRANSACTION HELP TEXT             PAGE 2
THE SIGNON SCREEN ALLOWS A TERMINAL USER TO GAIN ACCESS TO THE SAMP APPLICATION
 SYSTEM.
DATA ENTRY IS REQUIRED FOR THE FOLLOWING FIELDS:
   USERID   -  1 TO 6 CHARACTERS
   PROJECT  -  1 CHARACTER
   GROUP    -  1 CHARACTER
   LOCKWORD -  1 TO 8 CHARACTERS

DATA ENTRY IS OPTIONAL IN THE FOLLOWING FIELDS:
   OPTION   -  VALUES A,B,D,H,I ARE ALLOWED
   TRX      -  VALUE IS MXX, WHERE M IS TRANSACTION MODE AND XX IS THE
               TRANSACTION ID. THIS IS VALID ONLY IF OPTION D IS ALSO
               SELECTED. ENTRY OF TRX WILL DISPLAY THE PRIMARY KEY
               SELECTION SCREEN.
   KEY      -  IF OPTION 'D' AND TRX 'MXX' ARE ALSO FILLED IN, THE NEXT
               SCREEN DISPLAYED IS THE SEGMENT DISPLAY SCREEN IF THE
               KEY VALUE WAS FOUND IN THE DATA BASE.
PRESS ENTER TO RETURN TO SIGNON SCREEN.$$


PRESS ENTER TO PROCESS. ACTION R1 TO RETURN TO PAGE 1.
   *** ENTER DATA FOR ADD ***
```

Figure 12-9. HT-HELP Generation Screen (Page 2)

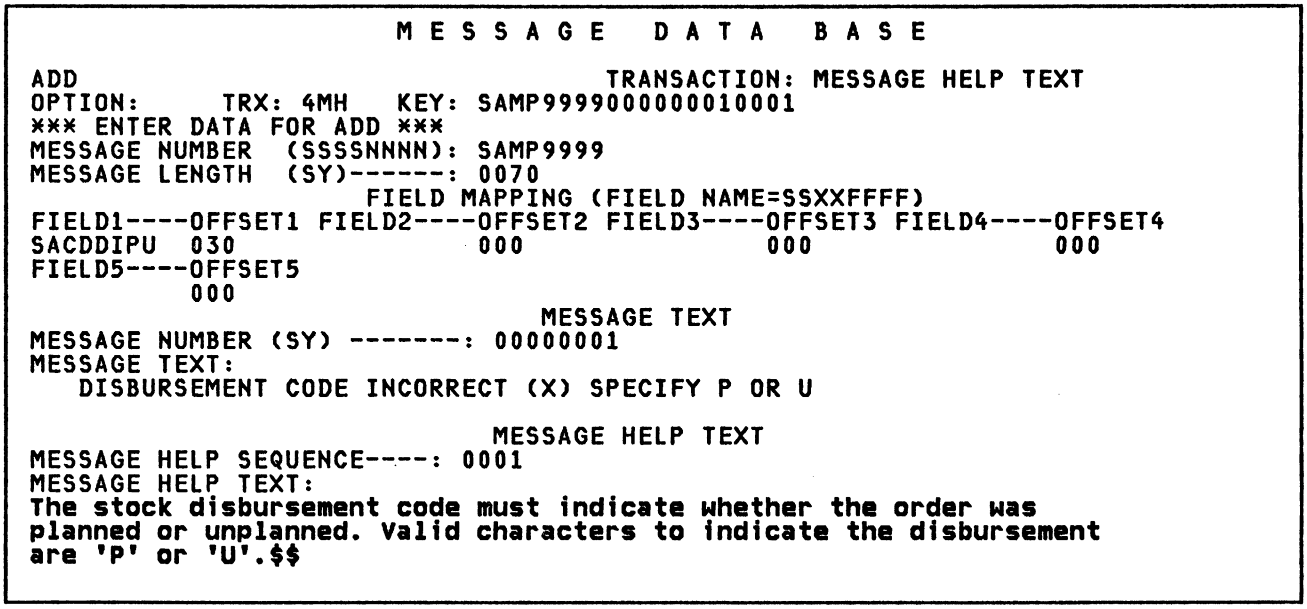
If an HT segment is not filled, the end of message characters should be specified after the last valid character. End of message characters are defined in installation time (DEFADF). The default is $$.

Transaction 'MH' creates the HELP text for the error or warning messages.

Error Message HELP Text (MH) (Segment Length = 250 bytes)

Position	Length	Data Type	Data Description
1	4	alphanum	segment sequence field (key)
5	9		reserved
14	79	alphanum	text that represents one line on the error/warning screen.
93	79	alphanum	text that represents one line on the error/warning screen.
172	79	alphanum	text that represents one line on the error/warning screen.

The 'MH' transaction screen allows for entry of up to 3 lines of HELP text. Multiple MH segments may be added.



```
                       M E S S A G E   D A T A   B A S E
ADD                                 TRANSACTION: MESSAGE HELP TEXT
OPTION:     TRX: 4MH   KEY: SAMP999900000010001
*** ENTER DATA FOR ADD ***
MESSAGE NUMBER  (SSSSNNNN): SAMP9999
MESSAGE LENGTH  (SY)------: 0070
                     FIELD MAPPING (FIELD NAME=SSXXFFFF)
FIELD1----OFFSET1 FIELD2----OFFSET2 FIELD3----OFFSET3 FIELD4----OFFSET4
SACDDIPU  030               000               000               000
FIELD5----OFFSET5
          000
                                MESSAGE TEXT
MESSAGE NUMBER (SY) -------: 00000001
MESSAGE TEXT:
   DISBURSEMENT CODE INCORRECT (X) SPECIFY P OR U

                             MESSAGE HELP TEXT
MESSAGE HELP SEQUENCE----: 0001
MESSAGE HELP TEXT:
The stock disbursement code must indicate whether the order was
planned or unplanned. Valid characters to indicate the disbursement
are 'P' or 'U'.$$
```

Figure 12-10. MH-Message HELP Generation Screen

CREATING HELP TEXT WITH BATCH TRANSACTIONS

Dynamic rules for HELP text can be entered using batch input. This may be more convenient for input of a large number of HELP screens. You may refer to "Batch Input of Dynamic Rules" on page 5-9 in Chapter 5, "Message Sending and Display" for a general discussion of updating the Message Data Base in batch. The batch input layouts for HELP text are:

HE - Screen HELP Header

Card	Column	Length	Description
1	9	8	Key of HE segment

Sample:
```
MFC1B2HESAMPESO@
```

HT - Screen HELP Text

Card	Column	Length	Description
1	9	8	Key of HE segment
	17	4	Key of HT segment
2	1	60	Header for HELP Screen
3	1	79	Text for Screen HELP
4	1	79	Text for Screen HELP
5	1	79	Text for Screen HELP
6	1	79	Text for Screen HELP
7	1	79	Text for Screen HELP
8	1	79	Text for Screen HELP
9	1	79	Text for Screen HELP
10	1	79	Text for Screen HELP
11	1	79	Text for Screen HELP
12	1	79	Text for Screen HELP
13	1	79	Text for Screen HELP
14	1	79	Text for Screen HELP
15	1	79	Text for Screen HELP
16	1	79	Text for Screen HELP
17	1	79	Text for Screen HELP
18	1	79	Text for Screen HELP
19	1	79	Text for Screen HELP
20	1	79	Text for Screen HELP
21	1	79	Text for Screen HELP
22	1	79	Text for Screen HELP

Sample:

```
MFC1B2HTSAMPESO@0001

THIS IS THE HEADER FOR THE SAMPLE SIGNON SCREEN HELP TEXT.

THIS IS LINE 1 OF THE HELP TEXT.

THIS IS LINE 2 OF THE HELP TEXT.

THIS IS THE LAST LINE OF THE HELP TEXT$$$$.
```

Note: If cards 4 through 22 are not needed the end of data characters should be entered twice.

MH - Message HELP Text

Card	Column	Length	Description
1	9	8	Key of HD segment
	17	8	Key of Last SY segment under this HD segment
	25	4	Key of MH segment
2	1	79	Text for Message HELP
3	1	79	Text for Message HELP
4	1	79	Text for Message HELP

Sample:

```
MFC1B4MHSAMP9999000000010001

THIS IS LINE 1 OF THE ERROR HELP TEXT.

THIS IS THE LAST LINE OF THE ERROR HELP TEXT$$$$.
```

Note: If cards 3 or 4 are not needed the end of data characters should be entered twice.

CHAPTER 13. NATIONAL LANGUAGE SUPPORT

OVERVIEW

IMSADF II supports the generation and execution of multi-language applications. The following languages are translated and provided as standard support:

- English
- French
- German
- Japanese
- Korean
- Portuguese
- Spanish
- Swedish

In addition, language components based on English are provided which may be modified for language requirements beyond the supplied languages. National Language Support (NLS) is visible to the terminal end user via screens and messages. It is not intended to cover all areas where the application developer is involved, such as static and dynamic rule definition.

At installation, a default language and optionally a set of alternate languages are specified for each ADFID. Each alternate language is associated with a SYSID. For each language specified at installation a complete set of system messages, base screens and Rules Generator screen definition modules are installed. These components are available to the application developer and the end user for NLS implementation. IMSADF II allows the same transaction code to execute across different SYSIDs. By assigning each SYSID to a different language the end user can view the transaction under the language of his choice.

DEVELOPING A NLS APPLICATION

Developing an application with multiple language support follows essentially the same steps as any standard application. Static and dynamic rules must be generated, messages must be created and screens must be built. The remainder of this section discusses only those areas which deviate from the normal generation definitions or online procedures.

Static Rule and Screen Generation

The main difference in this area is the requirement that each screen be shown with screen literals in the appropriate language. Literals are provided from two sources. First is the user defined field names derived from either the SNAME parameter or Screen Image. These are under the control of the developer and so require no special action.

The IMSADF II supplied literals on the other hand must be in the appropriate language. A separate Rules Generator module for each specific language is included at installation. A new keyword USRLANG is provided to define the appropriate language to the Rules Generator. Refer to the description of USRLANG in the IMS Application Development Facility II Version 2 Release 2 Application Development Reference. If USRLANG is not specified, the installation defined default language is used. USRLANG may be defined for the entire generation run in the SYSTEM statement or by transaction in the GENERATE statement.

Since the same transaction may be required to run under more than one language, a new keyword, ALIAS, is implemented to allow renaming of Segment Layout and Segment Handler rules under more than one SYSID. The standard naming convention of these rules is ssss---- where ssss is SYSID and ---- varies depending upon the rule. The ALIAS keyword on the GENERATE statement allows the rule to be renamed with different SYSIDs. Although this feature is implemented for NLS, it can also be used with transactions running under different SYSIDs which have the same language.

Following is an example of the generation of an application in two languages (German and French).

```
SYSTEM  SYSID=AAAA,.........,...

SEGMENT ID=01,...,...,...,...,...,...

FIELD   ID=ZZ01,...,...,...,...,...

FIELD   ID=ZZ02,...,...,...,...,...

FIELD   ID=ZZ03,...,...,...,...,...

FIELD   ID=ZZ04,...,...,...,...,...

GENERATE  OPTION=SGALL,SEGMENT=01,ALIAS=BBBB

GENERATE  OPTION=CVALL,TRXID=A1,...,...,USRLANG=G,SPOS=SIMAGE

.......... screen image  for German screen.................

.......... screen image  .................................

.......... screen image  .................................

GENERATE  OPTION=CVALL,TRXID=A1,USRLANG=F,SYSID=BBBB,

.......... screen image  for French screen.................

.......... screen image  .................................

.......... screen image  .................................
```

In this example SYSID AAAA is defined as a German application system and BBBB is a French application system. The first GENERATE builds Segment Layout and Segment Handler rules with the names:

AAAASR01	alias	BBBBSR01	Segment Layout rule
AAAAS01	alias	BBBBS01	Segment Handler rule

The second generate builds an Input Transaction Rule with the name AAAAPGA1 plus the MFS for the Segment Display and Primary Key Selection screens for German. The last generate builds an Input Transaction Rule with the name BBBBPGA1 plus the MFS for the Segment Display and Primary Key Selection screens for French.

Audit Logic Creation

Since the same transaction may execute under more than one SYSID, it is important to provide a technique which allows audit logic and edits to be written once and accessed from either language. This can be accomplished by writing the audit logic in subroutines which can be called from audits under each SYSID. For example; the transaction A1 generated above may have an audit requirement to check each field in segment 01 for valid data. If errors are found the appropriate error messages(s) should be displayed.

```
SUBNAME = SUBROUTINE000001

PARMNAME = BASE

PROCESS
```

```
PO

  IF AA01ZZ01 = 9999

    ERRORMSG = 0001

  ENDIF

  IF AA01ZZ02 = 9999

    ERRORMSG = 0002

  ENDIF

  RETURN
```

The subroutine tests fields ZZ01 and ZZ02 for valid data and sets the appropriate errors if the value is 9999. The subroutine is called from either the German audit (SYSID=AAAA) or the French audit (SYSID=BBBB).

```
SYSID = AAAA

AGROUP = 0001

SEGID = 01

FIELD = ZZ01

PROCESS

PO

  CALL 'SUBROUTINE000001'



SYSID = BBBB

AGROUP = 0001

SEGID = 01

FIELD = ZZ01

PROCESS

PO

  CALL 'SUBROUTINE000001'
```

User Message Creation

Error and Warning messages are created and maintained under the SYSID to which they apply. This is the same technique used in a single-language environment. If a multilingual transaction is implemented, the same message must be added to the Message Data Base under each SYSID, in the appropriate language. In the above example messages 0001 and 0002 must be added under the key AAAA0001,AAAA0002 for German and BBBB0001,BBBB0002 for French. Then, depending upon which language is in effect when the error is detected, the correct message will appear on the screen.

The system supplied messages are loaded during installation under the installed ADFID (default language) and each SYSID assigned to an alternate language. These SYSIDs may or may not coincide with the SYSIDs later associated with an application system. Each language will have one installed SYSID, which contains the supplied system messages, and as many application system SYSIDs as desired.

If in the above example the system was installed with a default language of German under ADFID of GERM and an alternate language of French under a SYSID of FREN the message would appear in the Message Data Base as follows:

• GERMxxxx	Supplied system messages in German
• FRENxxxx	Supplied system messages in FRENCH
• AAAAxxxx	User Error/Warning messages in German
• BBBBxxxx	User Error/Warning messages in French

User Written Sign-On Exit

The link between a SYSID and the appropriate language is under the control of the application developer through a user written Sign-on Lockword exit. This exit is required if more than one language is used under one ADFID.

The exit is responsible for setting a one-byte field in the SPA named SPAULANG. This field is set to the default or alternate language id associated with the current SYSID. In addition the SPASYSID field can be set depending upon input from one of the Sign-on screen fields. In either case it is the responsibility of the exit to maintain a correct SPASYSID, SPAULANG correspondence. If the exit does not set SPAULANG the default id is used. In the above example the exit should be written to set SPAULANG as follows:

Entered SYSID	Exit sets SPAULANG
AAAA	G
BBBB	F
FREN	F
GERM	G (or do not set)

Upon return from the exit IMSADF II will validate the language ID against those defined at installation and use it to retrieve the appropriate base screen formats. This allows all screens, IMSADF II supplied and user created, to display in the correct language.

During transaction execution, the display of an IMSADF II system error will cause a test to be made between SPAULANG and the installed default language ID. If they are not equal the ID in SPAULANG will be checked against the alternate language IDs to determine the correct language for the message.

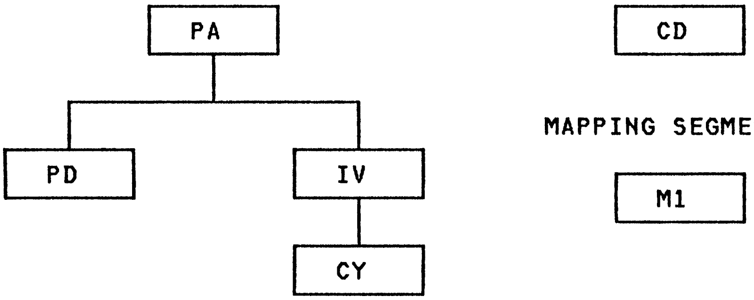
Note: The language ID set in the exit is NOT changed during a Project Group switch, since the exit is not invoked. The new application will therefore use the same language as the previous one.

Additional information on lockword exits and an example of an exit for multiple languages may be found under "Multiple National Languages" on page 9-9.

APPENDIX A. SAMPLE SYSTEM RULES GENERATOR STATEMENTS

The following source statements build the sample problem application system (SAMP), which is supplied with the product and forms the basis of all the examples that appear in this document.

```
***********************************************************************
*       APPLICATION DEFINITION INPUT STATEMENTS FOR PARTS DATA BASE
***********************************************************************

SYSTEM   SYSID=SAMP,DBID=PA,                        RULE ID CHARS
         USRLANG=E,                                 ENGLISH
         SOMTX=OR,                       DEFAULT SECONDARY OPTION CODE
         PCBNO=1,                        PCB NUMBER FOR DATA BASE
         SHEADING='S A M P L E   P R O B L E M',    GENERAL HEADING
         SFORMAT=DASH,                              SCREEN FORMAT
         PGROUP=ZZ                                  PROJECT GROUP
GENERATE OPTION=SIGN,                    REQUEST CONVERSATIONAL SIGNON
         DEVNAME=(2),
         DEVCHRS=(0),
         DEVTYPE=(2)
GENERATE OPTION=POM,POMENU=ABCDFHI       REQUEST PRIMARY OPTION MENU

***********************************************************************
* PARTS DATA BASE                        PSEUDO SEGMENT
```



```
***********************************************************************
*       APPLICATION DEFINITION INPUT FOR ROOT SEGMENT
***********************************************************************

 SEGMENT LEVEL=1,ID=PA,NAME=PARTROOT,LENGTH=50,
         SKSEG=18        PART SEGMENT
   FIELD ID=KEY,LENGTH=17,POS=1,KEY=YES,NAME=PARTKEY,
         SNAME='PART NUMBER',DISP=YES
   FIELD ID=DESC,LENGTH=20,POS=27,SNAME='DESCRIPTION',DISP=YES,REL=YES

***********************************************************************
*       APPLICATION DEFINITION INPUT FOR STANDARD INFORMATION SEGMENT
***********************************************************************

 SEGMENT ID=PD,PARENT=PA,NAME=STANINFO,LENGTH=85
*        SNAME='STANDARD INFORMATION'
   FIELD ID=KEY,LENGTH=2,POS=1,KEY=YES,NAME=STANKEY,DISP=NO,
         SNAME='KEY FIELD'
   FIELD ID=PRCD,LENGTH=2,POS=19,SNAME='PROC CODE',DISP=YES
   FIELD ID=INVC,LENGTH=1,SNAME='INVENTORY CODE',DISP=YES
   FIELD ID=PLRV,LENGTH=2,SNAME='PLAN REV NO',DISP=YES
   FIELD ID=MKDP,LENGTH=4,POS=48,SNAME='MAKE DEPT',DISP=YES
   FIELD ID=COMM,LENGTH=4,POS=54,SNAME='COMM CODE',DISP=YES
   FIELD ID=RISP,LENGTH=2,POS=62,TYPE=DEC,SNAME='RIGHT MAKE TIME',
         DISP=YES,SLENGTH=2
   FIELD ID=WRSP,LENGTH=2,POS=71,TYPE=DEC,SNAME='WRONG MAKE TIME',
         DISP=YES,SLENGTH=2
```

```
***********************************************************************
*        APPLICATION DEFINITION INPUT FOR INVENTORY SEGMENT
***********************************************************************

 SEGMENT ID=IV,PARENT=PA,NAME=STOKSTAT,KEYNAME=STOCKEY,LENGTH=160,
*        SNAME='INVENTORY',
         SKLEFT='INVENTORY         UNIT      CURRENT ',
         SKLEFT='LOCATION          PRICE     REQMNTS ',
         SKRIGHT='  ON      TOTAL      DISBURSEMENTS  ',
         SKRIGHT=' ORDER    STOCK   PLANNED  UNPLANNED'
   FIELD ID=W,LENGTH=2,POS=1,KEY=YES,SNAME='00',DISP=NO,
         COL=1,SLENGTH=2
   FIELD ID=AREA,LENGTH=1,KEY=YES,SNAME='AREA',DISP=YES,COL=3,REQ=NO
   FIELD ID=INVD,LENGTH=2,KEY=YES,SNAME='INV DEPT',DISP=YES,COL=4,
         REQ=NO
   FIELD ID=PROJ,LENGTH=3,KEY=YES,SNAME='PROJECT',DISP=YES,COL=6,REQ=NO
   FIELD ID=DIV,LENGTH=2,KEY=YES,SNAME='DIVISION',DISP=YES,COL=9,REQ=NO
   FIELD ID=FILL,LENGTH=6,KEY=YES,SNAME='FILLER',DISP=NO,COL=11,REQ=NO
   FIELD ID=PRIC,LENGTH=9,POS=21,TYPE=DEC,DEC=2,SLENGTH=9,
         SNAME='UNIT PRICE',DISP=YES
   FIELD ID=RPRI,LENGTH=7,POS=23,TYPE=DEC,DEC=2,RELATED=YES,COL=19,
         DISP=NO
   FIELD ID=UNIT,LENGTH=4,POS=35,SNAME='UNIT',DISP=YES
   FIELD ID=COAP,LENGTH=3,POS=51,TYPE=DEC,SNAME='ATTR COAP',DISP=YES,
         SLENGTH=3
   FIELD ID=PLAN,LENGTH=3,TYPE=DEC,POS=54,
         SNAME='ATTR PLANNED',DISP=YES,SLENGTH=3
   FIELD ID=COAD,LENGTH=1,POS=57,SNAME='ATTR COAD',DISP=YES
   FIELD ID=CDAY,LENGTH=3,SNAME='STOCK DATE',DISP=YES,POS=72,
         SLENGTH=3
   FIELD ID=TDAY,LENGTH=3,TYPE=DEC,SNAME='LAST TRANS',DISP=YES,
         SLENGTH=3
   FIELD ID=REQC,LENGTH=7,POS=90,TYPE=DEC,SLENGTH=7,
         SNAME='RQMNTS CURRENT',DISP=YES
   FIELD ID=RREQ,LENGTH=5,POS=92,TYPE=DEC,RELATED=YES,COL=29,DISP=NO
   FIELD ID=REQU,LENGTH=7,TYPE=DEC,POS=98,SLENGTH=7,
         SNAME='RQMNTS UNPLAN',DISP=YES
   FIELD ID=ONOR,LENGTH=7,POS=106,TYPE=DEC,SNAME='ON ORDER',
         DISP=YES,SLENGTH=7
   FIELD ID=RONO,LENGTH=5,POS=108,TYPE=DEC,RELATED=YES,COL=38,DISP=NO
   FIELD ID=STCK,LENGTH=7,TYPE=DEC,POS=114,SLENGTH=7,
         SNAME='TOTAL STOCK',DISP=YES
   FIELD ID=RSTC,LENGTH=5,POS=116,TYPE=DEC,RELATED=YES,COL=47,DISP=NO
   FIELD ID=DIPL,LENGTH=7,TYPE=DEC,POS=122,SLENGTH=7,
         SNAME='DISB PLAN',DISP=YES
   FIELD ID=RDIP,LENGTH=5,POS=124,TYPE=DEC,RELATED=YES,COL=56,DISP=NO
   FIELD ID=DIUN,LENGTH=7,TYPE=DEC,POS=130,SLENGTH=7,
         SNAME='DISB UNPLAN',DISP=YES
   FIELD ID=RDIU,LENGTH=5,POS=132,TYPE=DEC,RELATED=YES,COL=65,DISP=NO
   FIELD ID=DISP,LENGTH=7,TYPE=DEC,POS=138,SLENGTH=7,
         SNAME='DISB SPARES',DISP=YES
   FIELD ID=DIDV,LENGTH=7,TYPE=DEC,POS=146,SLENGTH=7,
         SNAME='DISB DIVERS',DISP=YES

***********************************************************************
*        APPLICATION DEFINITION INPUT FOR CYCLE COUNT SEGMENT
***********************************************************************

 SEGMENT ID=CY,PARENT=IV,NAME=CYCCOUNT,LENGTH=25
*        SNAME='CYCLE COUNT'
   FIELD ID=KEY,LENGTH=2,POS=1,KEY=YES,NAME=CYCLKEY,
         SNAME='20',DISP=NO,REQ=YES
   FIELD ID=CNTA,LENGTH=7,TYPE=DEC,SNAME='PHYS COUNT',SLENGTH=7
   FIELD ID=STCK,LENGTH=7,TYPE=DEC,POS=11,SLENGTH=7,
         SNAME='BOOK COUNT'
```

```
***********************************************************************
*        PSEUDO SEGMENT FOR SPECIAL PROCESSING
***********************************************************************

  SEGMENT ID=CD,TYPE=PS        'CLOSE/DISBURSE INVENTORY'
    FIELD ID=CLOR,L=7,TYPE=DEC,SNAME='CLOSE ORDER',DISP=YES,SL=7
    FIELD ID=CLST,L=7,TYPE=DEC,SNAME='STOCK INCR',DISP=YES,SL=7
    FIELD ID=DQTY,L=7,TYPE=DEC,SNAME='DISBURSE QTY',DISP=YES,SL=7
    FIELD ID=DIPU,L=1,AUDIT=YES,SNAME='PLANNED/UNPLAN',DISP=YES

***********************************************************************
*        MAPPING SEGMENT FOR SPECIAL PROCESSING
***********************************************************************

  SEGMENT ID=M1,TYPE=MAP
    FIELD ID=CLOR,LEN=4,TYPE=BIN,SEGID=CD
    FIELD ID=CLST,LEN=4,TYPE=BIN,SEGID=CD
    FIELD ID=DQTY,LEN=4,TYPE=BIN,SEGID=CD
    FIELD ID=ONOR,LEN=4,TYPE=BIN,SEGID=IV
    FIELD ID=STCK,LEN=4,TYPE=BIN,SEGID=IV
    FIELD ID=DIPL,LEN=4,TYPE=BIN,SEGID=IV
    FIELD ID=DIUN,LEN=4,TYPE=BIN,SEGID=IV
    FIELD ID=DIPU,LEN=1,SEGID=CD

***********************************************************************
*        GENERATE RULES FOR STANDARD SEGMENT PROCESSING
***********************************************************************

  GENERATE SEGMENT=(PA,PD,IV,CY),OPTIONS=(SEGL,SEGH)
  GENERATE TRXID=PA,DBPATH=PA,OPTIONS=CVALL,
           TRXNAME='PART SEGMENT',
           DEVNAME=(2),
           DEVCHRS=(0),
           DEVTYPE=(2)
  GENERATE TRXID=PD,DBPATH=PD,OPTIONS=CVALL,
           TRXNAME='STANDARD INFORMATION',
           DEVNAME=(2),
           DEVCHRS=(0),
           DEVTYPE=(2)
  GENERATE TRXID=IV,DBPATH=IV,OPTIONS=CVALL,
           TRXNAME='INVENTORY',
           DEVNAME=(2),
           DEVCHRS=(0),
           DEVTYPE=(2)
  GENERATE TRXID=CY,DBPATH=CY,OPTIONS=CVALL,
           TRXNAME='CYCLE COUNT',
           DEVNAME=(2),
           DEVCHRS=(0),
           DEVTYPE=(2)

***********************************************************************
*        GENERATE RULES FOR SPECIAL PROCESSING
***********************************************************************

  GENERATE SEGMENT=(M1,CD),OPTIONS=SEGL
  GENERATE TRXID=CD,TSEG=(CD,PD),DBPATH=IV,BYPASS=YES,LANGUAGE=ASMINT,
           OPTIONS=CVALL,SPECIAL=YES,KEYSL=YES,
           TRXNAME='CLOSE/DISBURSE INVENTORY',
           DEVNAME=(2),
           DEVCHRS=(0),
           DEVTYPE=(2)

***********************************************************************
*        GENERATE SECONDARY OPTION MENU
***********************************************************************

  GENERATE OPTIONS=SOM,UPDATE=N,
           LINKREQ=YES                LINK-EDIT PRECEDING RULES    1.3
  GENERATE OPTION=STLE,PGMID=OR,LINKLIB=PGMLOAD
  GENERATE OPTION=SPLE,PGMID=CD,LANGUAGE=ASMINT,MAPTABLE=(M1),
           AEXIT=USERAUDT,
           SHTABLE=(PD,IV),LINKLIB=PGMLOAD
```

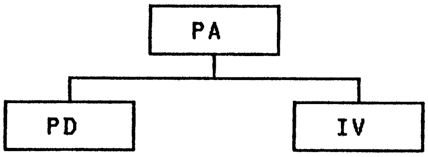
APPENDIX B. ALTERNATE TWIN PROCESSING TECHNIQUES

Chapter 6, "Complex Transactions" gave guidelines for implementing twin processing applications using enhanced automatic twin processing functions. This appendix contains static rules, dynamic rules, and audit exit (in COBOL) for an alternate, user-written method of processing twin segments. These rules are intended to illustrate techniques rather than provide a complete application.

STATIC RULES

The static rules given illustrate the need to define aliases of the segment IV that is to be retrieved and updated in multiple occurrences on a single Data Display screen. These aliases are named I1, I2, I3, I4, and I5. The first three are actually displayed and are therefore in the screen image; the last two are used as working storage by the logic of the audit rules given later.

```
// EXEC MFC1G

***********************************************************************
*   APPLICATION DEFINITION INPUT STATEMENTS FOR PARTS DATA BASE
***********************************************************************
SYSTEM   SYSID=SAMP,DBID=PA,                                RULE ID CHARS
         SOMTX=OR,                               DEFAULT SECONDARY OPTION CODE
         PCBNO=1,                                PCB NUMBER FOR DATA BASE
         SHEADING='S A M P L E   P R O B L E M',            GENERAL HEADING
         SFORMAT=DASH,                                      SCREEN FORMAT
         PGROUP=ZZ                                          PROJECT GROUP

***********************************************************************
* PARTS DATA BASE
*
*
*
*
*
*
*
*
```



```
***********************************************************************
*   APPLICATION DEFINITION INPUT FOR ROOT SEGMENT
***********************************************************************

 SEGMENT LEVEL=1,ID=PA,NAME=PARTROOT,LENGTH=50,
         SKSEG=18         PART SEGMENT
   FIELD ID=KEY,LENGTH=17,POS=1,KEY=YES,NAME=PARTKEY,
         SNAME='PART NUMBER',DISP=YES,REL=YES
   FIELD ID=DESC,LENGTH=20,POS=27,SNAME='DESCRIPTION',DISP=YES,REL=YES

***********************************************************************
*   APPLICATION DEFINITION INPUT FOR INVENTORY SEGMENT
***********************************************************************

 SEGMENT ID=I1,PARENT=PA,NAME=STOKSTAT,LENGTH=160
 FIELD   ID=ILOC,KEY=YES,LENGTH=16,NAME=STOCKEY,
         AUDIT=YES      THE AUDIT LOOP CHECKS OTHER TWINS ALSO
 FIELD   ID=PRIC,SLENGTH=10,LENGTH=9,POS=21,TYPE=DEC,DEC=2
 FIELD   ID=REQC,LENGTH=7,POS=90,TYPE=DEC
 FIELD   ID=STCK,LENGTH=7,POS=114,TYPE=DEC
 SEGMENT ID=I2,PARENT=PA,NAME=STOKSTAT,LENGTH=160
 FIELD   ID=ILOC,KEY=YES,LENGTH=16,NAME=STOCKEY
 FIELD   ID=PRIC,SLENGTH=10,LENGTH=9,POS=21,TYPE=DEC,DEC=2
 FIELD   ID=REQC,LENGTH=7,POS=90,TYPE=DEC
 FIELD   ID=STCK,LENGTH=7,POS=114,TYPE=DEC
 SEGMENT ID=I3,PARENT=PA,NAME=STOKSTAT,LENGTH=160
 FIELD   ID=ILOC,KEY=YES,LENGTH=16,NAME=STOCKEY
 FIELD   ID=PRIC,SLENGTH=10,LENGTH=9,POS=21,TYPE=DEC,DEC=2
 FIELD   ID=REQC,LENGTH=7,POS=90,TYPE=DEC
 FIELD   ID=STCK,LENGTH=7,POS=114,TYPE=DEC
 SEGMENT ID=I4,PARENT=PA,NAME=STOKSTAT,LENGTH=160
 FIELD   ID=ILOC,KEY=YES,LENGTH=16,NAME=STOCKEY
 FIELD   ID=PRIC,SLENGTH=10,LENGTH=9,POS=21,TYPE=DEC,DEC=2
 FIELD   ID=REQC,LENGTH=7,POS=90,TYPE=DEC
 FIELD   ID=STCK,LENGTH=7,POS=114,TYPE=DEC
 SEGMENT ID=I5,PARENT=PA,NAME=STOKSTAT,LENGTH=160
 FIELD   ID=ILOC,KEY=YES,LENGTH=16,NAME=STOCKEY
 FIELD   ID=PRIC,SLENGTH=10,LENGTH=9,POS=21,TYPE=DEC,DEC=2
 FIELD   ID=REQC,LENGTH=7,POS=90,TYPE=DEC
 FIELD   ID=STCK,LENGTH=7,POS=114,TYPE=DEC

***********************************************************************
*     GENERATE RULES FOR TWIN PROCESSING
***********************************************************************

  SEGMENT ID=TW,TYPE=PS
    FIELD ID=FLAG,LENGTH=1,PAUDIT=YES,AUDIT=YES,FAUDIT=YES,MSG=YES
    FIELD ID=SKEY,LENGTH=17         SAVED PARENT KEY
    FIELD ID=CKEY,LENGTH=33,POS=19  CONCATENATED KEY
    FIELD ID=PKEY,LENGTH=17,POS=19  PARENT KEY
    FIELD ID=DKEY,LENGTH=16,POS=36  DEPENDENT SEG KEY
    FIELD ID=FKEY,LENGTH=16         KEY OF FIRST SEGMENT (FOR POSN.)
    FIELD ID=KEY1,LENGTH=16         SAVED KEYS
    FIELD ID=KEY2,LENGTH=16
    FIELD ID=KEY3,LENGTH=16

  GENERATE TRXID=IN,DBPATH=PA,OPT=CVALL,
           TRXNAME='INVENTORIES',SPOS=SIMAGE,DLET=(I1,I2,I3),
           TSEGS=(TW,I1,I2,I3,I4,I5),CURSOR=FLAG
&=1
                          'INVENTORY INFORMATION
&=2
&SYSMSG
REQUEST: &5FLAG
OPTION:  &OPT TRX: &TRAN KEY: &KEY
&=1
  PART NUMBER:  &5KEY.PA            DESCRIPTION:  &6DESC.PA
&=1
  =====================================================================
&=1
  INVENTORY             UNIT             REQUIREMENTS        TOTAL
  LOCATION              PRICE            CURRENT             STOCK
 &5ILOC.I1             &5PRIC.I1         &5REQC.I1         &5STCK.I1
 &5ILOC.I2             &5PRIC.I2         &5REQC.I2         &5STCK.I2
 &5ILOC.I3             &5PRIC.I3         &5REQC.I3         &5STCK.I3
&ENDS

***********************************************************************

  GENERATE OPTIONS=SGALL
  GENERATE OPTIONS=CVSYS
  GENERATE OPTIONS=SOMSS
// EXEC MFSUTL,COND.S1=(0,LT)
//S1.SYSIN DD DSN=&&MFS,DISP=(OLD,DELETE)
//
```

HIGH LEVEL AUDIT LANGUAGE STATEMENTS

As suggested in Chapter 4, "The Auditor and the Audit Data Base," a NOP (no operation) is included in the P1 part of the PRELIM phase to avoid possible problems of mixing with the PROCESS phase.

The logic of the first section is as follows:

- Retrieve the first segment with a GUU call. The code at label RETRIEVE is also invoked via a branch out of the PROCESS phase when the user enters the "R" (Retrieve) request.

- At label NEXTSEGS, the following occurrences are retrieved with GN calls. That label can also be reached by a branch out of the PROCESS phase when the user enters the "M" (More) request or when the segments must be re-retrieved for a redisplay to the user.

- Finally, the I4 occurrence is retrieved so that the user can be told whether more occurrences are present.

HIGH LEVEL AUDIT LANGUAGE STATEMENTS

```
* TWIN PROCESSING
* PRE-AUDIT PHASE - RETRIEVE SEGMENTS STARTING WITH GUU CALL
          SYSID   = SAMP
          AGROUP  = YYYY
          SEGID   = TW
          FIELD   = FLAG
          PRELIM
          P1
          NOP
          P2
* FIRST GET PARENTS CONCATENATED KEY
          SATWPKEY = SAPAKEY
* SAVE IT
          SATWSKEY = SAPAKEY
* GUU CALL IS ISSUED DURING PRE-AUDIT.
* IT IS ALSO ISSUED DURING UPDATE PASSES IF USER
* ENTERS "R" OR IF FIRST SEGMENT DISPLAYED HAS BEEN DELETED
RETRIEVE: IF GUU SATWPKEY I1 OK
            SATWKEY1 = SAI1ILOC
          ELSE
            SATWKEY1 = '               '
            GOTO ENDRETR
          ENDIF
* RETRIEVE THE SECOND AND SUBSEQUENT SEGMENTS
* BEGIN BY SAVING THE FIRST KEY FOR LATER POSITIONING
NEXTSEGS: SATWFKEY = SAI1ILOC
          SETTWIN = 'I2,I3'
          SETARRAY = SATWKEY2
          DOTWIN = 1 TO 2
            IF GN SATWPKEY I2 OK
              SATWKEY2 = SAI2ILOC
            ELSE
              SATWKEY2 = '               '
              GOTO ENDRETR
            ENDIF
          ENDTWIN
* THIS LAST IS A CHECK FOR MORE SEGMENTS THAN CAN BE SHOWN ON 1 SCREEN
  IF GN SATWPKEY I4 OK
            SPAERMSG = 'ENTER REQUEST: "M" TO VIEW MORE INVENTORY'
          ELSE
            GOTO ENDRETR
          ENDIF
* NO MORE TO RETRIEVE - TERMINATE UNLESS UNEXPECTED DL/I STATUS CODE
ENDRETR:  IF STATCODE ¬= 'GE,  '
            GOTO DLIERROR
          ENDIF
          FLAG = '_'
```

The PROCESS phase code to perform the updates is placed in the message leg (P2) because it must be performed only after the validation in the field audit leg (P1) has been completed.

First, we insert the root segment if the user has invoked mode 4. In this mode, multiple occurrences can be inserted under the root at the same time as the root. Next, the generalized updates against the first twin I1 are performed. This code allows for three cases of changing the key, detected by comparing the entered key (SAI1ILOC) with the saved key (SATWKEY1).

- Where the user blanks an existing key, the segment will be deleted.

- Where the user alters an existing key, the old segment will be deleted and the same data inserted under the new key.

- Where the user enters a key into a previously blank area, the new segment is inserted.

```
          PROCESS
          P2
          IF MODE = 4
*   MUST INSERT ROOT SEGMENT BEFORE INSERTING DEPENDENTS
            IF ISRT IMMED
            KEYFIELD PA NOT OK
              GOTO DLIERRUP
            ENDIF
          ENDIF
          SATWPKEY = SATWSKEY
* UPDATE CYCLE - REPEATED AS NECESSARY.
* LOGIC IS TO HANDLE DELETIONS AND INSERTIONS; REPLACE CALLS
* ARE PERFORMED BY THE SEGUPDTE CALL FROM AUDIT EXIT 71
          SETTWIN = 'I1,I2,I3'
          SETARRAY = SATWKEY1
          DOTWIN = 1 TO 3
*   IF USER HAS ALTERED KEY OF SEGMENT, MUST BE A DLET AND/OR ISRT
            IF SATWKEY1 ¬= SAI1ILOC
*     IF SEGMENT KEY WAS NON-BLANK BEFORE DISPLAY,
*     MUST BE A COPY (DLET + ISRT) OR A STRAIGHT DLET.
              IF SATWKEY1 ¬= '                '
                IF HDEL IMMED KEYFIELD I1 NOT OK
                  GOTO DLIERRUP
                ENDIF
              ENDIF
*     IF USER HAS ENTERED NON-BLANK KEY, INSERT SEGMENT
              IF SAI1ILOC ¬= '                '
                IF ISRT IMMED SATWPKEY I1 NOT OK
                  GOTO DLIERRUP
                ENDIF
*         RESET SAVED FIRST KEY FOR POSITIONING IF THIS ONE IS LOWER
                IF SAI1ILOC < SATWFKEY
                  SATWFKEY = SAI1ILOC
                ENDIF
              ENDIF
            ENDIF
          ENDTWIN
```

Now we call an audit exit routine that invokes the IMSADF II service routine SEGUPDTE. It will update those segments that need to be replaced. It will not do anything to the segments that have already been inserted or deleted since their flags will have been reset by those operations. If an unexpected DL/I status code is encountered by SEGUPDTE, it will issue a ROLL call to undo all changes made in this IMS/VS transaction scheduling.

Next we re-initialize the segment areas and re-retrieve the segments. We can cope with the user requesting "M" or "R" at the same time as updates are performed.

```
* CALL SEGUPDTE TO PERFORM REPLACE CALLS
          IF AEXIT 71 RETURN = FALSE
            GOTO DLIERRUP
          ENDIF
* GET CONCATENATED KEY OF LAST SEGMENT
          SATWPKEY = SATWSKEY
          SATWDKEY = SAI3ILOC
* NOW RESET SEGMENT AREAS PRIOR TO RE-RETRIEVAL
          INITSEGS = 'I1,I2,I3'
          SATWKEY1 = '                '
          SATWKEY2 = '                '
          SATWKEY3 = '                '
* IF USER HAS REQUESTED MORE SEGMENTS, POSITION THE PCB ON THE LAST
* SEGMENT PREVIOUSLY DISPLAYED
          IF FLAG = 'M'
            IF GU SATWCKEY I4 OK
              NOP
            ENDIF
*   THE FOLLOWING MESSAGE WILL BE OVERLAID BY THE ENTER M MESSAGE
*   UNLESS WE REACH THE END OF THE TWIN CHAIN
```

```
            SPAERMSG = 'ENTER REQUEST R: TO RETURN TO THE FIRST DISPLAY'
*   RETRIEVE NEXT SEGMENT, WHICH IS TO BE DISPLAYED NOW
            SATWPKEY = SATWSKEY
            IF GN SATWPKEY I1 OK
              SATWKEY1 = SAI1ILOC
            ELSE
              SATWKEY1 = '                '
*     IF THERE IS NO NEXT SEGMENT, ENSURE RE-POSITIONING WILL BE AT
*     THE END OF THE TWIN CHAIN UNLESS THE USER MAKES AN INSERTION
              SATWFKEY = '9999999999999999'
              GOTO ENDRETR
            ENDIF
            GOTO NEXTSEGS
          ENDIF
* IF USER REQUESTS RETURN TO FIRST DISPLAY, GO AND ISSUE GUU CALL
          IF FLAG = 'R'
            GOTO RETRIEVE
          ENDIF
* OTHERWISE DISPLAY FROM FIRST OCCURRENCE ON PREVIOUS DISPLAY
* SAVED IN FKEY AND POSSIBLY AMENDED BY INSERTIONS OF LOWER KEY VALUE
          SATWPKEY = SATWSKEY
          SATWDKEY = SATWFKEY
          IF GU SATWCKEY I1 OK
            SATWKEY1 = SAI1ILOC
            GOTO NEXTSEGS
          ENDIF
* IF FIRST OCCURRENCE NOT FOUND, TRY NEXT
          SATWPKEY = SATWSKEY
          SATWDKEY = SATWFKEY
          IF GN SATWCKEY I1 OK
            SATWKEY1 = SAI1ILOC
            GOTO NEXTSEGS
          ENDIF
* IF THIS FAILS, PERFORM GUU
          SATWPKEY = SATWSKEY
          GOTO RETRIEVE
* ALWAYS ISSUE ROLL CALL IF ANY DL/I ERRORS
DLIERROR: NOP
DLIERRUP: NOP
         ROLLCALL = 'UNEXPECTED DATA BASE ERROR. CONTACT SYSTEM SUPPORT'
ENDIT:    NOP
```

The following illustrates that the validation, consisting of a check that we are not inserting a duplicate key, can be applied to every occurrence even though only the first twin's key is marked AUDIT=YES in the Rules Generator input.

```
*
          ADFID   = MFC1
          SYSID   = SAMP
          AGROUP  = YYYY
          SEGID   = I1
          FIELD   = ILOC
          PROCESS
          P1
* PERFORM VALIDITY CHECKING FOR ALL TWIN SEGMENT ALIASES
          SETTWIN = 'I1,I2,I3'
          SETARRAY = SATWKEY1
          DOTWIN = 1 TO 3
*   IF KEY CHANGED AND NON-BLANK, CHECK VALIDITY
            IF SATWKEY1 ¬= ILOC
              IF ILOC ¬= '                '
*       IF DUPLICATE KEY, SEND ERROR MESSAGE
                SATWDKEY = ILOC
                IF GU SATWCKEY I5 OK
                  ERRORMSG = 9801
                ENDIF
              ENDIF
            ENDIF
          ENDTWIN
```

ERROR MESSAGE

The error message need be defined only once for the twins because the special value VARLIST5 can be used to put the value of the field in error into the message. This will contain the appropriate twin occurrence.

```
// EXEC MFC1B
//TRANSIN DD *
* AUDIT ERROR MESSAGES
MFC1B1HD
SAMP9801
MFC1B2HD
SAMP98010070VARLIST5024
MFC1B4SYSAMP9801
00000001FOR INVENTORY LOCATION                  THERE IS ALREADY AN ENTRY
//
```

AUDIT EXIT

The following COBOL audit exit routine invokes the IMSADF II service routine SEGUPDTE to replace those segments that have been changed. This saves having to write the data base replace (REPL) calls in the high level audit language and overcomes the limitation that it is impossible, in the audit language, to test whether a segment has been changed.

Normally, the SEGUPDTE service routine would be called by the transaction driver after all the audits have been completed. It would be too late in this case, however, since the audit rules cause the segments to be re-retrieved after the updates.

```
//EXSUB EXEC COBUC,
// PARM.COB=(NORES,NODYN,NOENDJOB,LIB,APOST,NOSEQ,
//       'SIZE=450000,BUF=128K')
//COB.SYSLIN DD DSN=IMSADF3.PROGRAM.OBJ(AUDTEXIT),DISP=OLD
//COB.SYSLIB DD DSN=IMSADF3.MACLIB.ASM,DISP=SHR
//COB.SYSIN DD *
       IDENTIFICATION DIVISION.
        PROGRAM-ID.
           UPDTEXIT.
        DATE-COMPILED. JULY 20,1982.
        REMARKS.
           AUDIT EXIT ROUTINE TO UPDATE DATA BASES.
        ENVIRONMENT DIVISION.
         CONFIGURATION SECTION.
          SOURCE-COMPUTER. IBM-370.
          OBJECT-COMPUTER. IBM-370.
        DATA DIVISION.
        WORKING-STORAGE SECTION.
        77 DD                 PICTURE XX.
        77 GUU                PICTURE X(4) VALUE 'GUU '.
        77 FE                 PICTURE XX VALUE 'FE'.
        77 FALSE              PICTURE X VALUE LOW-VALUES.
        01 TRUTH1             PICTURE 9(4) COMP VALUE 128.
        01 TRUTH2             REDEFINES TRUTH1.
            03 FILLER         PICTURE X.
            03 TRUE           PICTURE X.
        01 DATA-DESC.
            03 KEY-FIELD      PICTURE X(4).
            03 LEFT-PAREN     PICTURE X(1).
            03 SEGID          PICTURE X(2).
            03 COMMA-POS1     PICTURE X(1).
            03 FLAG           PICTURE X(1).
            03 COMMA-POS2     PICTURE X(1).
            03 SETTING        PICTURE X(1).
            03 RIGHT-PAREN    PICTURE X(1).
            03 FILLER         PICTURE X(28).
        LINKAGE SECTION.
        77 AUDITED-FIELD      PICTURE X(17).
        77 FIELD-DESC         PICTURE X.
        77 AUDIT-DESC         PICTURE XX.
        77 AUDIT-PCB          PICTURE X.
        77 COMOPT             PICTURE X.
        77 TRUE-FALSE         PICTURE X.
        77 FUNCTION-INDIC     PICTURE X.
        77 PCBLIST            PICTURE X.
        77 COKEY              PICTURE X.
        77 RELATED-FIELD      PICTURE X(255).
        77 RELATED-FIELD-DESC PICTURE X.
        COPY SPACOBOL.
        PROCEDURE DIVISION USING AUDITED-FIELD,
            FIELD-DESC, AUDIT-DESC,
            AUDIT-PCB, COMOPT, TRUE-FALSE, FUNCTION-INDIC, SPADSECT,
            PCBLIST, COKEY, RELATED-FIELD, RELATED-FIELD-DESC.
            IF AUDIT-DESC NOT = '71' THEN GOBACK.
            CALL 'SEGUPDTE'.
            MOVE TRUE TO TRUE-FALSE.
            IF SPARTNCD > 0 MOVE FALSE TO TRUE-FALSE.
            IF SPARTNCD = 16 MOVE TRUE TO TRUE-FALSE.
            GOBACK.
// EXEC MFC1G
//G1.SCREENS DD DISP=(,DELETE)
//G1.SYSLIB DD
//     DD
//     DD DSN=SYS1.COBLIB,DISP=SHR
  SYSTEM SYSID=SAMP
  GENERATE OPTIONS=STLE,PGMID=OR,AEXIT=UPDTEXIT
//
```

APPENDIX C. REPORT WRITING EXAMPLE

Chapter 10, "Batch Processing" gives guidelines for creating output and reports. This appendix contains the static and dynamic rules for that example, and a sample of the results.

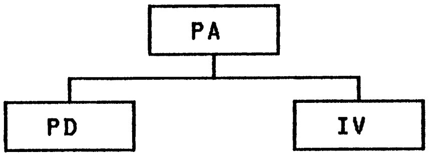
These rules are provided merely as examples of how to implement report writing using IMSADF II. They are intended to illustrate techniques rather than provide a complete application.

Because the sample requires the IMS/VS-supplied PARTS data base, the sample problem is not applicable to a CICS/DB2 installation. It may be invoked only as a BMP in the IMS/DB2 environment.

STATIC RULES

The static rules given illustrate the need to define segments with TYPE=OUT for formatting headings and detail lines.

```
// EXEC MFC1G

************************************************************************
*   APPLICATION DEFINITION INPUT STATEMENTS FOR PARTS DATA BASE
************************************************************************

SYSTEM   SYSID=SAMP,DBID=PA,                         RULE ID CHARS
         SOMTX=OR,                        DEFAULT SECONDARY OPTION CODE
         PCBNO=1,                         PCB NUMBER FOR DATA BASE
         SHEADING='S A M P L E   P R O B L E M',    GENERAL HEADING
         SFORMAT=DASH,                               SCREEN FORMAT
         PGROUP=ZZ                                   PROJECT GROUP

************************************************************************
* PARTS DATA BASE
```



```
*
*
*
*
*
*
*
*

************************************************************************
*   APPLICATION DEFINITION INPUT FOR ROOT SEGMENT
************************************************************************

 SEGMENT LEVEL=1,ID=PA,NAME=PARTROOT,LENGTH=50,
         SKSEG=18        PART SEGMENT
   FIELD ID=KEY,LENGTH=17,POS=1,KEY=YES,NAME=PARTKEY,
         SNAME='PART NUMBER',DISP=YES,REL=YES
   FIELD ID=DESC,LENGTH=20,POS=27,SNAME='DESCRIPTION',DISP=YES,REL=YES

************************************************************************
*   APPLICATION DEFINITION INPUT FOR INVENTORY SEGMENT
************************************************************************

 SEGMENT ID=IV,PARENT=PA,NAME=STOKSTAT,KEYNAME=STOCKEY,LENGTH=160,
*        SNAME='INVENTORY',
         SKLEFT='INVENTORY          UNIT      CURRENT ',
         SKLEFT='LOCATION           PRICE     REQMNTS ',
         SKRIGHT='  ON       TOTAL      DISBURSEMENTS  ',
         SKRIGHT=' ORDER     STOCK   PLANNED  UNPLANNED'
   FIELD ID=W,LENGTH=2,POS=1,KEY=YES,SNAME='00',DISP=NO,
         COL=1,SLENGTH=2
   FIELD ID=AREA,LENGTH=1,KEY=YES,SNAME='AREA',DISP=YES,COL=3,REQ=NO
   FIELD ID=INVD,LENGTH=2,KEY=YES,SNAME='INV DEPT',DISP=YES,COL=4,
         REQ=NO
   FIELD ID=PROJ,LENGTH=3,KEY=YES,SNAME='PROJECT',DISP=YES,COL=6,REQ=NO
   FIELD ID=DIV,LENGTH=2,KEY=YES,SNAME='DIVISION',DISP=YES,COL=9,REQ=NO
   FIELD ID=FILL,LENGTH=6,KEY=YES,SNAME='FILLER',DISP=NO,COL=11,REQ=NO
   FIELD ID=PRIC,LENGTH=9,POS=21,TYPE=DEC,DEC=2,SLENGTH=9,
         SNAME='UNIT PRICE',DISP=YES
   FIELD ID=RPRI,LENGTH=7,POS=23,TYPE=DEC,DEC=2,RELATED=YES,COL=19,
         DISP=NO
   FIELD ID=UNIT,LENGTH=4,POS=35,SNAME='UNIT',DISP=YES
   FIELD ID=COAP,LENGTH=3,POS=51,TYPE=DEC,SNAME='ATTR COAP',DISP=YES,
         SLENGTH=3
   FIELD ID=PLAN,LENGTH=3,TYPE=DEC,POS=54,
         SNAME='ATTR PLANNED',DISP=YES,SLENGTH=3
   FIELD ID=COAD,LENGTH=1,POS=57,SNAME='ATTR COAD',DISP=YES
   FIELD ID=CDAY,LENGTH=3,SNAME='STOCK DATE',DISP=YES,POS=72,
         SLENGTH=3
   FIELD ID=TDAY,LENGTH=3,TYPE=DEC,SNAME='LAST TRANS',DISP=YES,
         SLENGTH=3
   FIELD ID=REQC,LENGTH=7,POS=90,TYPE=DEC,SLENGTH=7,
         SNAME='RQMNTS CURRENT',DISP=YES
   FIELD ID=RREQ,LENGTH=5,POS=92,TYPE=DEC,RELATED=YES,COL=29,DISP=NO
   FIELD ID=REQU,LENGTH=7,TYPE=DEC,POS=98,SLENGTH=7,
         SNAME='RQMNTS UNPLAN',DISP=YES
   FIELD ID=ONOR,LENGTH=7,POS=106,TYPE=DEC,SNAME='ON ORDER',
         DISP=YES,SLENGTH=7
   FIELD ID=RONO,LENGTH=5,POS=108,TYPE=DEC,RELATED=YES,COL=38,DISP=NO
   FIELD ID=STCK,LENGTH=7,TYPE=DEC,POS=114,SLENGTH=7,
         SNAME='TOTAL STOCK',DISP=YES
   FIELD ID=RSTC,LENGTH=5,POS=116,TYPE=DEC,RELATED=YES,COL=47,DISP=NO
   FIELD ID=DIPL,LENGTH=7,TYPE=DEC,POS=122,SLENGTH=7,
         SNAME='DISB PLAN',DISP=YES
   FIELD ID=RDIP,LENGTH=5,POS=124,TYPE=DEC,RELATED=YES,COL=56,DISP=NO
   FIELD ID=DIUN,LENGTH=7,TYPE=DEC,POS=130,SLENGTH=7,
         SNAME='DISB UNPLAN',DISP=YES
   FIELD ID=RDIU,LENGTH=5,POS=132,TYPE=DEC,RELATED=YES,COL=65,DISP=NO
   FIELD ID=DISP,LENGTH=7,TYPE=DEC,POS=138,SLENGTH=7,
         SNAME='DISB SPARES',DISP=YES
   FIELD ID=DIDV,LENGTH=7,TYPE=DEC,POS=146,SLENGTH=7,
         SNAME='DISB DIVERS',DISP=YES

************************************************************************
*   PSEUDO SEGMENT FOR REPORT WRITING
************************************************************************
*
  SEGMENT ID=RP,TYPE=PS
    FIELD ID=FLAG,L=1,DISP=NO,AFA=YES USED AS AN AUDIT FIELD
    FIELD ID=PAGE,L=3,DISP=NO,TYPE=PD PAGE NUMBER
    FIELD ID=LINE,L=3,DISP=NO,TYPE=PD LINE NUMBER
    FIELD ID=TOTS,L=5,DISP=NO,TYPE=PD TOTAL STOCK

************************************************************************
*   OUTPUT FORMAT RULES FOR HEADERS AND REPORT DETAIL LINE
************************************************************************
*
  SEGMENT ID=H1,TYPE=OUT  FIRST HEADER
    FIELD TEXT='1         ',LEN=10
    FIELD TEXT='REPORT ON PART NU',LEN=17
    FIELD TEXT='MBERS AND ',LEN=10
    FIELD TEXT='STOCK LEVELS IN THE ',LEN=20
    FIELD TEXT='SAMPLE DATA BASE    ',LEN=20
    FIELD TEXT='                PAGE',LEN=20
    FIELD ID=PAGE,LEN=5,SEGID=RP
  SEGMENT ID=H2,TYPE=OUT  SECOND HEADER
    FIELD TEXT='0         ',LEN=10
    FIELD TEXT='PART NUMBER      ',LEN=17
    FIELD TEXT='          ',LEN=10
    FIELD TEXT='DESCRIPTION         ',LEN=20
    FIELD TEXT='INVENTORY LOCATION  ',LEN=20
    FIELD TEXT='TOTAL STOCK',LEN=11
  SEGMENT ID=H3,TYPE=OUT  THIRD HEADER
    FIELD TEXT='          ',LEN=10
    FIELD TEXT='-----------      ',LEN=17
    FIELD TEXT='          ',LEN=10
    FIELD TEXT='-----------         ',LEN=20
    FIELD TEXT='------------------  ',LEN=20
    FIELD TEXT='-----------',LEN=11
  SEGMENT ID=DL,TYPE=OUT  PART NUMBER DETAIL LINE
    FIELD TEXT='          ',LEN=10
    FIELD ID=KEY,LEN=17,SEGID=PA
    FIELD TEXT='          ',LEN=10
    FIELD ID=DESC,LEN=20,SEGID=PA
  SEGMENT ID=DI,TYPE=OUT  INVENTORY DETAIL LINE
    FIELD TEXT='                    ',LEN=20
    FIELD TEXT='                 ',LEN=17
    FIELD TEXT='                    ',LEN=20
    FIELD ID=AREA,LEN=1,SEGID=IV
    FIELD ID=INVD,LEN=2,SEGID=IV
    FIELD ID=PROJ,LEN=3,SEGID=IV
    FIELD ID=DIV,LEN=2,SEGID=IV
    FIELD TEXT='            ',LEN=12
    FIELD ID=STCK,LEN=11,SEGID=IV
  SEGMENT ID=TL,TYPE=OUT  INVENTORY TOTAL LINE
    FIELD TEXT='                    ',LEN=20
    FIELD TEXT='                 ',LEN=17
    FIELD TEXT='                    ',LEN=20
    FIELD TEXT='                  ',LEN=20
    FIELD ID=TOTS,LEN=11,SEGID=RP
  SEGMENT ID=TM,TYPE=OUT  INVENTORY TOTAL MARKER
    FIELD TEXT='                    ',LEN=20
    FIELD TEXT='                 ',LEN=17
    FIELD TEXT='                    ',LEN=20
    FIELD TEXT='                  ',LEN=20
    FIELD TEXT='===========',LEN=11
    EJECT

*************************************************************************
*   GENERATE RULES FOR BATCH STANDARD SEGMENT PROCESSING
*************************************************************************

  GENERATE OPT=SGALL
*
* GENERATE BATCH INPUT TRANSACTION RULE
*
  GENERATE TRXID=RP,TSEGS=(RP,PA,IV),
         OPTIONS=BAIT,
         STX=(TRX,H1),STX=(TRX,H2),STX=(TRX,H3),STX=(TRX,DL),
         STX=(TRX,TM),STX=(TRX,TL),STX=(TRX,DI)
*
* GENERATE SAMPBD01 LOAD MODULE
*
  GENERATE OPTIONS=BDLE,PGMID=01,OFRTABLE=(H1,H2,H3,DL,TM,TL,DI),
         SHTABLE=(PA,IV),ITTABLE=RP,LDRULE=YES,SLRTABLE=(RP),
         PHEAD='BATCH REPORT WRITER EXAMPLE'
```

HIGH LEVEL AUDIT LANGUAGE CODING

The logic consists of a loop through the root segments (PA) with a nested loop through the inventory segments (IV). A subtotal is printed of the stock levels of each part. Line and page counts are used to control pagination. The above segments of TYPE=OUT each begin with an ASA control character to cause spacing and page ejection.

```
SYSID = SAMP
AGROUP = YYYY
SEGID = RP
FIELD = FLAG
PROCESS
P0
* LOOP READING PA SEGMENTS FROM THE SAMPLE DATA BASE,
* PRINTING THEM USING SEND IMMED UNTIL A STATUS CODE OF
* GB (END OF DATA BASE) IS FOUND.
SARPPAGE = 0 SARPLINE = 50
SARPTOTS = 0
SWITCH1 = ON
* THE LOOP THROUGH PA SEGMENTS IS TERMINATED BY THE EXIT STATEMENT
DO WHILE SWITCH1 = ON
  IF GN KEYFIELD PA NOT OK
    IF STATCODE = 'GB'
      EXIT
    ELSE
      ERRORMSG = 9999
    ENDIF
  ENDIF
  IF SARPLINE >= 50
*   PAGE THROW AND HEADERS
    SARPLINE = 0 SARPPAGE = SARPPAGE + 1
    SEND IMMED 'SAORH101'
    SEND IMMED 'SAORH201'
    SEND IMMED 'SAORH301'
  ENDIF
* PRINT DETAIL LINE
  SEND IMMED 'SAORDL01'
  SARPLINE = SARPLINE + 1
  SWITCH2 = ON
  COUNTER1 = 0
* LOOP THROUGH DEPENDENT SEGMENTS (IV)
* LOOP TERMINATED WHEN SWITCH2 = OFF
* DEPENDENT SEGMENTS COUNTED IN COUNTER1
  DO WHILE SWITCH2 = ON
    IF GN SAPAKEY IV NOT OK
      IF STATCODE = 'GB,GE'
        SWITCH2 = OFF
*       END OF IV SEGMENTS.
*       IF THERE WERE IV SEGMENTS, SHOW TOTAL AND RESET IT.
        IF COUNTER1 > 0
          SEND IMMED 'SAORTM01'
          SEND IMMED 'SAORTL01'
          SEND IMMED 'SAORTM01'
          SARPLINE = SARPLINE + 3
          SARPTOTS = 0 COUNTER1 = 0
        ENDIF
      ELSE
        ERRORMSG = 9999
      ENDIF
    ELSE
*     PRINT INVENTORY DETAIL LINE AND MAINTAIN TOTAL
      SEND IMMED 'SAORDI01'
      SARPLINE = SARPLINE + 1
      COUNTER1 = COUNTER1 + 1
      SARPTOTS = SARPTOTS + SAIVSTCK
    ENDIF
  ENDDO
ENDDO
```

SAMPLE OUTPUT

Sample output for transaction SAMPB5RP is shown in Figure C-1.

```
REPORT ON PART NUMBERS AND STOCK LEVELS IN THE SAMPLE DATA BASE   PAGE 1
PART NUMBER          DESCRIPTION         INVENTORY LOCATION  TOTAL STOCK
-----------          -----------         ------------------  -----------
02AN960C10           WASHER
                                          AA16511                    126
                                          AK2877F                     88
                                         28009126                    680
                                                             ===========
                                                                     894
                                                             ===========
02CK05CW181K         CAPACITOR
                                          VF52906                      0
                                         25900326                    660
                                         25910926                      8
                                                             ===========
                                                                     668
                                                             ===========
02CSR13G104KL        KR1J50KS
                                          DB7455R                     14
                                          SK21713                      4
                                         25502526                     14
                                                             ===========
                                                                      32
                                                             ===========
02JAN1N976B          DIODE CODE-A
                                         25509126                     17
                                                             ===========
                                                                      17
                                                             ===========
02MS16995-28         SCREW
                                          AA16511                     30
                                          BA16515                      8
                                          FF5546D                     43
                                         25910926                    100
                                                             ===========
                                                                     181
                                                             ===========
02N51P3003F000       SCREW
                                         25906026                    360
                                                             ===========
                                                                     360
                                                             ===========
02RC07GF273J         RESISTOR
                                          AK24527                     33
                                         28009126                     17
                                         28011126                     26
                                                             ===========
                                                                      76
                                                             ===========
02106B1293P009       RESISTOR
                                         25900326                   1055
                                         25906026                      0
                                         25910926                    320
                                                             ===========
                                                                    1375
                                                             ===========
```

Figure C-1. Sample Output Page

APPENDIX D. APPLICATION IMPLEMENTATION

The recommendations in this appendix relate to activities that precede and succeed application development. These recommendations must not be regarded as rigid. They must be evaluated for their applicability in each situation.

TRANSACTION AND SCREEN DESIGN

In order to maximize programmer productivity with IMSADF II and deliver function to the user as quickly as possible, the application prototyping capability of IMSADF II should be exploited to the fullest. This entails a somewhat different phasing of design and development activities, making use of a "master rules" concept.

Figure D-1 illustrates how an operational application system, capable of data base inquiries and maintenance and providing full updating capabilities, can be implemented on the basis of a completed data base design. Details on how to do this are presented in the first chapters of this manual. The more advanced functions described in later chapters can then be used for the complex application requirements.

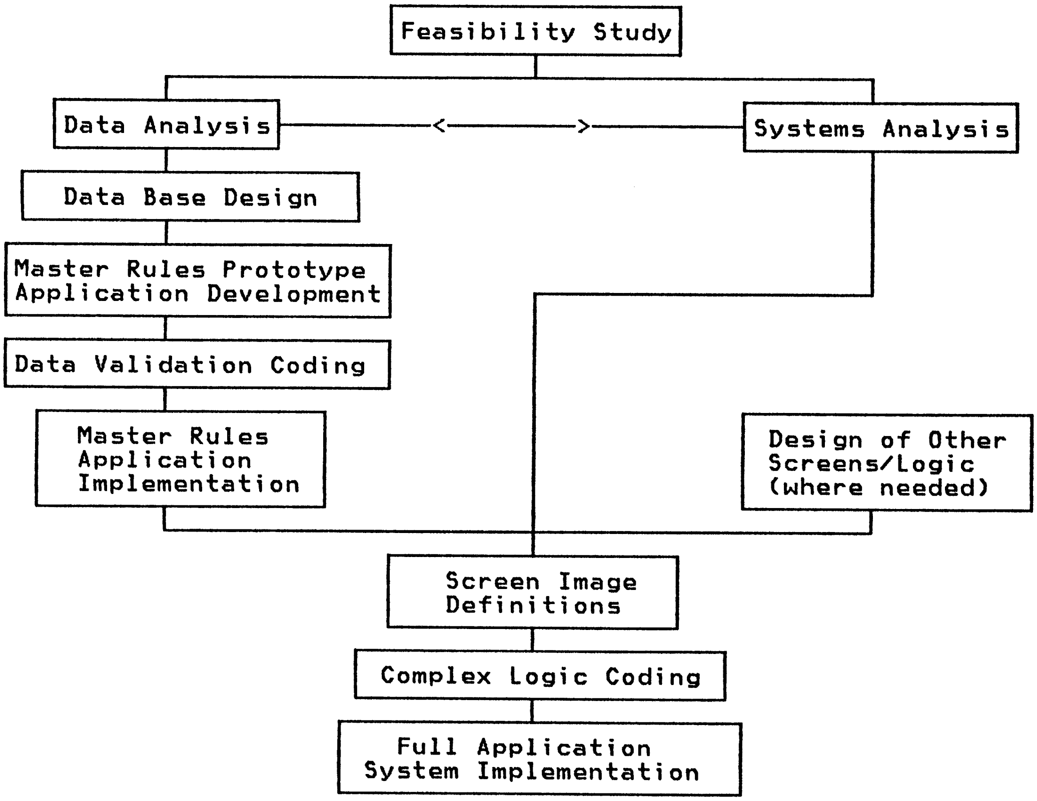

Feasibility Study
Data Analysis
Systems Analysis
Data Base Design
Master Rules Prototype Application Development
Data Validation Coding
Master Rules Application Implementation
Design of Other Screens/Logic (where needed)
Screen Image Definitions
Complex Logic Coding
Full Application System Implementation

Figure D-1. Suggested Ordering of Design and Development Activities

Figure D-2 shows the recommended structure of these components.

Component	**Recommended Structure**
Segment definitions	For each data base segment. a member should be created containing FIELD statements that define field ID, key attributes, position, length, data type and SNAME.
Master rules	There should be one member in a PDS containing the master rules for an application system. It should: • reference the segment definitions by means of INCLUDE statements • define one transaction per data base segment, using default screens • request the Sign-On and Primary Option Menu screens
Sign-on and security profiles	Profiles should be maintained by the person responsible for security.
Complex transactions and screen images	Any transaction not in the master rules is considered complex for the purposes of this discussion. It is recommended that one source statement member be maintained in a PDS for every complex transaction. The member will contain: • a SYSTEM statement • a GENERATE statement that defines the transaction (with the TRXID option) • a screen image or reference (IMAGE keyword) to a screen image • SEGMENT statements referring to any segments in the master rules that are used by this transaction, together with INCLUDE statements • any FIELD statements needed to override field definitions included as a result of INCLUDE statements • any pseudo segment or alias segment definitions • a GENERATE OPT=SOMSS statement

Figure D-2 (Part 1 of 2). Recommended Structure of Development Components

Component	Recommended Structure
High level audit language statements	If there are any audit rules that apply to a field wherever it is used in an application system (generally validation rules), these should be kept in one member. All other audit rules are best viewed as part of a transaction. All audit operations that apply to a transaction should be coded in a member that is named in association with that transaction. A suggested format for the name is: **ssssAAtx** where: **ssss** is the system ID **AA** is a literal **tx** is the transaction ID It is suggested that this same name be used for the audit group and that a separate audit group code be used for each transaction. This means that AGROUP = AAtx should be coded at the start of each member after SYSID = ssss. AGROUP=AAtx must also be coded on the corresponding Rules Generator statement, GENERATE TRXID=tx.
Messages and message rules	These should be separated by transaction, just as audit rules are separated, with a member containing systemwide messages.

Figure D-2 (Part 2 of 2). Recommended Structure of Development Components

EASE OF MAINTENANCE

To ensure that application systems are maintainable, it is essential that several components be kept in step. These components are most easily maintained in source statement form. They are:

- Rules Generator statements
- high level audit language statements
- messages and message rules in the form in which they can be submitted to the IMSADF II batch utility
- sign-on and security profiles

The decision left open in the above discussion is whether to allow auditing in the master rules. IMSADF II permits you to distinguish between audit rules that validate a field wherever it is used and audit rules that control application logic. The above recommendations allow systemwide auditing of a field to be associated with the master rules. Such an approach will be appropriate in some installations and not in others.

NAMING CONVENTIONS

IMSADF II already has extensive naming conventions. The only purpose of applying further conventions is to avoid clashes in usage of the same name. In large projects where several individuals are developing parts of an application system, conventions are needed to allocate:

- transaction IDs
- IDs for pseudo segments, mapping segments, aliases, and twins
- message numbers
- automatic message headers

IADF has naming conventions for all the components listed above. If your installation is not using IADF, it is suggested that a simple convention on name and number range be adopted. The project leader should allocate ranges of the above IDs and numbers to individual developers.

MOVING FROM TEST TO PRODUCTION

It is recommended that IMSADF II application systems be transferred from test to production at the source statement level. IADF's migration feature can be used to do this. If the recommendations given in "Ease of Maintenance" on page D-3 are followed there will be a set , of three source statement members for each complex transaction plus the security profiles. The three source member types are:

- Rules Generator source statements
- high level audit language statements
- messages and message rules in batch input form

For static rules, move all source statements and members of the INCLUDE libraries into a production source library and rerun the Rules Generator on the production system.

For dynamic rules, transfer the high level audit language statements into the production source library and rerun the compiler to load the audit rules in the Audit Data Base. Maintain and transfer the message rules in source form as submitted to the batch utility (JCL procedure MFC1B) and transfer this source file also from test to production. Then rerun the batch utility on the production system to load them on to the Message Data Base. Do the same to load the tables in the Audit Data Base. Finally, run the utility to make the audit rules static (see "Static Audit Rules" on page D-5).

If the security profiles are the same in test and production, they could be transferred in the same way. Normally, however, these would be different and would be set up separately on the production system by the data base administrator.

During subsequent application maintenance, the smallest unit of transfer between test and production would be the set of three source statement members that define a transaction.

LOAD MODULE TRANSFER

If source level transfer is impractical in your installation, then load module transfer of rules produced by the Rules Generator is possible. The person responsible must be aware of which rules load modules are required to make up a transaction so that it is possible to check for completeness.

A static audit rule load module can be transferred in the same way as other load modules. Messages and message rules, however, must still be transferred in source form unless a user-written utility program can be made available to copy from the Message Data Base.

Particular care must be exercised to ensure that the Secondary Option Menu Rule load module does not get out of step if two members of a development project attempt to implement new transactions on the same day. It would be appropriate for the person responsible for security profiles to maintain the Secondary Option Menu Rule as well.

USING MULTIPLE IMSADF II SYSTEMS

The simple way to install IMSADF II and to separate test from production is to have IMS/VS separately installed for test and production (on the same or separate CPUs) and install IMSADF II separately on each IMS/VS installation.

If this cannot be done and the test and production systems are to reside on the same installation of IMS/VS, then the multiple IMSADF II capability, documented in the IMS Application Development Facility II

Version 2 Release 2 Application Development Reference, will be needed to ensure separation of the test IMSADF II from the production IMSADF II. Installing IMSADF II in this way is not normally the responsibility of those who develop applications, but there are certain effects that must be allowed for.

Each separate IMSADF II will be given a separate ADFID. A possible standard is to use the ADFID of MFC1 for testing and MFC2 for production. Then the production JCL procedure names will all begin with MFC2 instead of MFC1; the input to the batch driver for the IMSADF II dynamic rules data bases will likewise begin with MFC2 instead of MFC1. An additional statement must be placed in front of the SYSID statement to the high level audit language compiler. In our case, for the production system, that would be:

```
ADFID = MFC2
```

The default value for ADFID is MFC1.

To ensure separation, the installer of IMSADF II will use IMS/VS capabilities to schedule the test and production versions into separate IMS/VS message processing regions. In addition, the Rules Generator SYSTEM statement operands MFSTRLR and TRXTRLR must be coded on the production version of the application's static rules. MFSTRLR and TRXTRLR are documented in the IMS Application Development Facility II Version 2 Release 2 Application Development Reference. See the IMS Application Development Facility II Version 2 Release 2 Installation Guide for complete details on installing multiple IMSADF II systems.

STATIC AUDIT RULES

When an application system is in production, it is not necessary to have the Auditor retrieve audit rules from the data base every time a transaction is used. A utility is provided to read a set of audit rules from the data base and combine them into a single load module which can be held in the static rules library. Hence, the Auditor need load only a single member to obtain all the rules needed in the transaction, rather than performing many DL/I calls. (It is possible to eliminate even the load operation by using the preload facility documented in the IMS Application Development Facility II Version 2 Release 2 Application Development Reference.)

Here is an example of running the utility, with JCL:

```
//UTILITY EXEC ????B
//CARDOUT DD DSM=&&CARDS,UNIT=SYSDA,DISP=(NEW,PASS),
//           SPACE=(TRK,(1,1))
//TRANSIN DD  *
MFC1B6AM SAMPYYYY BANKYYYY
/*
//RULESGEN EXEC MFCG1
//SYSIN    DD  DSN=&&CARDS,DISP=(OLD,DELETE)
```

Note: ???? is the installed ADFID (the default is MFC1).

Note the format of the control card: a code MFC1B6AM followed by a space, followed by up to six 8-byte identifiers of the form:

SSSSAAAA

where:

SSSS is the application system ID
AAAA is the audit group code

If more than six identifiers are needed, code more cards, each beginning MFC1B6AM. The end product, after running the Rules Generator procedure (MFC1G), is a set of load modules named with the 8-byte identifiers. If key audits are created using KNAME=ALT, the above applies.

When audit rules for key auditing are stored in the Audit Data Base under root keys that begin KEYAUDIT, a single, installation-wide load module named KEYAUDIT will be produced from the static audit rules utility. The control card to be submitted to the MFC1B procedure in this case would be:

```
  MFC1B6AM KEYAUDIT
```

By running the first JCL step and directing the CARDOUT data set to the printer, a useful listing of the audit rules may be obtained.

The Auditor will obtain rules from the data base unless directed to use a load module by coding the LRULE=YES operand on the GENERATE statement for a transaction or on the SYSTEM statement (to the Rules Generator). If the default audit group code (YYYY) is used, there will be one static audit rule per application system. In large, complex applications separate audit group codes - and hence separate - rules for different transactions or groups of transactions will simplify application maintenance.

Subroutines and tables can also be in static form. If coding in a static audit rule refers to tables or subroutines, these should also be in static audit rules (not necessarily in the same load module) in order to obtain best performance. However, if the Auditor cannot find a static audit rule load module, and LRULE=ALT was coded on the GENERATE statement, it will look for the appropriate rule on the audit data base.

APPENDIX E. SWITCHING BETWEEN COBOL AND IMSADF II TRANSACTIONS

It is possible to cause an IMSADF II transaction to switch to a non-IMSADF II IMS/VS conversational message processing programming (MPP). Such a program can also switch to an IMSADF II transaction. This is done with the IMS/VS "conversational program-to-program switch."

SWITCHING FROM IMSADF II TO COBOL

This is achieved by link-editing the non-IMSADF II MPP with the same name as an IMSADF II transaction mini-driver. An IMSADF II transaction switch will then cause an IMS/VS conversational program-to-program switch to the MPP. The IMSADF II transaction switch can be initiated by a user at the terminal altering the TRXID area on the Data Display screen or under the control of audit rules or a special processing program.

The sample COBOL programs (MPPs) shown below are link-edited to fit in with the sample application system (SAMP). They are associated with the IMSADF II transaction IDs SW and RS and therefore are named:

ssssTcc

where:

ssss is the application system ID
T is a literal
cc is the cluster code (SOMTX operand value)

The following GENERATE statements are necessary to cause these two TRXIDs to be entered into the Secondary Option Menu Rule:

```
GENERATE TRXID=SW,DBPATH=PA,SOMTX=SW,OPTION=INTR
GENERATE TRXID=RS,DBPATH=PA,SOMTX=RS,OPTION=INTR
GENERATE OPTIONS=SOM
```

In addition, the SW and RS TRXIDs must be added to the Sign-On Profile Authority segment (PR) with entries such as **SW53RS53**, where the final 3 in each 4-byte entry indicates that these TRXIDs are <u>not</u> to appear on the Secondary Option Menu screen.

The following high level audit language statements cause IMS/VS to switch to the programs named SAMPTSW or SAMPTRS.

```
SYSID  = SAMP
AGROUP = YYYY
SEGID = PA
FIELD = KEY
PRELIM
P1
IF TRXID = 'PA'
 TRXID = 'SW'
 ENDIF
IF TRXID = 'CY'
 TRXID = 'RS'
 ENDIF
```

Note: PAUDIT=YES must be coded on the FIELD statement that defines the KEY field in the PA root segment.

SWITCHING FROM COBOL TO IMSADF II

The COBOL programs (MPPs) shown below are named so that they can receive control from an IMSADF II transaction. They are both designed to switch to an IMSADF II transaction through the IMS/VS conversational program-to-program switch.

There are two techniques:

- Switching to the sign-on transaction
- Switching directly to the driver

SWITCHING TO THE SIGN-ON TRANSACTION

This is the simpler of the two techniques. It has some disadvantages:

- No information can be transmitted to the target IMSADF II transaction in the SPA communication area (SPAFLDSG) for use by an audit exit or special processing routine.

- Two IMS/VS message switches will take place to get to the IMSADF II Data Display screen. This is a performance overhead.

The technique is to pass to the sign-on transaction (????T01 where ???? is the ADFID) a message that looks to IMSADF II like a Sign-On screen entered at a user terminal. The COBOL program must supply the IMSADF II application system ID, the user ID, and the project/group code. In addition it can supply the lockword, OPTION, TRXID, and KEY fields. The program must also set the data portion of the SPA to binary zeros.

The layout of the message is as follows:

LL	(2)	Message length (92)
ZZ	(2)	Binary zero
USERID	(6)	User ID
PGROUP	(2)	Project/group
LOCKWORD	(8)	Lockword
FILLER	(8)	Spaces
SYSID	(4)	Application system ID
FILLER	(4)	Spaces
NLINES	(2)	Number of lines on screen (in character form, e.g., 24)
OPTION	(1)	Option (e.g., D)
TRXID	(3)	IMSADF II transaction ID (e.g., 5PD)
KEY	(50)	Concatenated key

The following program is intended as an illustration of how to use the technique.

```
IDENTIFICATION DIVISION.
PROGRAM-ID.
    SAMPTRS.
AUTHOR.
    IMSADF II.
DATE-WRITTEN.
    1982.
REMARKS.
        THIS SKELETON PROGRAM SWITCHES TO THE IMSADF II SIGNON
    SCREEN VIA THE IMS/VS MESSAGE QUEUE, THUS SIMULATING A USER
    SIGNING ON. IN THE MESSAGE IT PASSES, IT SETS THE IMSADF II
    APPLICATION SYSTEM ID, THE USER ID, PROJECT/GROUP, OPTION,
    TRXID AND CONCATENATED KEY FIELDS.  THE SPA IS SET TO LOW
    VALUES TO SIMULATE THE START OF A CONVERSATION.
ENVIRONMENT DIVISION.
CONFIGURATION SECTION.
SOURCE-COMPUTER.
    IBM-370.
OBJECT-COMPUTER.
    IBM-370.
INPUT-OUTPUT SECTION.
FILE-CONTROL.
```

```
DATA DIVISION.
FILE SECTION.
WORKING-STORAGE SECTION.
77  GU PIC XXXX VALUE 'GU  '.
77  ISRT PIC XXXX VALUE 'ISRT'.
77  CHNG  PIC XXXX VALUE 'CHNG'.
77  DISP-STAT PIC X(12) VALUE SPACES.
77  LT-SYS PIC X(12) VALUE 'LT-SYS-ERR.'.
77  DA-SYS PIC X(12) VALUE 'DA-SYS-ERR.'.
77  ALT-SYS PIC X(12) VALUE 'ALT-SYS-ERR.'.
COPY SPACOBOL.
  03  FILLER        PIC X(29967).
01  SPA-START REDEFINES SPADSECT.
  03  FILLER        PIC X(12).
  03  SPA-INIT      PIC X(31755).
01  MSG.
  03  MSG-LL        PIC 9(4) COMP VALUE 92.
  03  MSG-ZZ        PIC 9(4) COMP VALUE 0.
  03  MSG-USERID    PIC X(6).
  03  MSG-PGROUP    PIC X(2).
  03  MSG-LOCKWORD  PIC X(8).
  03  FILLER        PIC X(8) VALUE SPACES.
  03  MSG-SYSID     PIC X(4).
  03  FILLER        PIC X(4) VALUE SPACES.
  03  MSG-NLINES    PIC X(2) VALUE '24'.
  03  MSG-OPTION    PIC X(1).
  03  MSG-TRXID     PIC X(3).
  03  MSG-KEY       PIC X(50) VALUE SPACES.
LINKAGE SECTION.
01  LT-PCB.
    02  LTERM  PIC X(8).
    02  FILLER PIC XX.
    02 LT-STATUS PIC XX.
88  Q-EMPTY VALUE 'QC'.
88  LT-OK VALUE '  '.
    02 FILLER PIC X(12).
01  ALT-PCB1.
    02  FILLER PIC X(10).
    02 ALT-STAT1 PIC XX.
88  AQ-EMPTY1 VALUE 'QC'.
88  ALT-OK1 VALUE '  '.
    02 FILLER PIC X(12).
01  ALT-PCB2.
    02  FILLER PIC X(10).
    02 ALT-STAT2 PIC XX.
88  AQ-EMPTY2 VALUE 'QC'.
88  ALT-OK2 VALUE '  '.
    02 FILLER PIC X(12).
01  ADFWK-PCB.
    02 FILLER PIC X(10).
    02  ADFW-STATUS PIC XX.
88  ADF-NOTFND VALUE 'GE'.
88  ADFW-OK VALUE '  '.
PROCEDURE DIVISION.
    ENTRY 'DLITCBL' USING LT-PCB, ALT-PCB1, ALT-PCB2, ADFWK-PCB.
RE-START.
*    NOTE ***  RETRIEVE SPA FROM IMS/VS.
     CALL 'CBLTDLI' USING GU LT-PCB SPADSECT.
     IF Q-EMPTY GO TO RE-EXIT.
     IF NOT LT-OK MOVE LT-SYS TO DISP-STAT GO TO DISPLY-STAT.
     DISPLAY 'RESTART PROGRAM EXECUTING'.
*            APPLICATION CODING FINISHES HERE.
```

```
PREP-SWITCH.
*    NOTE ***  NOW PREPARE SPA AND MESSAGE FOR SWITCH TO ADF.
     MOVE LOW-VALUES   TO SPA-INIT.
     MOVE 'MFC1T01 '   TO SPATRANS.
     MOVE 'SAMP'       TO MSG-SYSID.
     MOVE '999999'     TO MSG-USERID.
     MOVE 'ZZ'         TO MSG-PGROUP.
     MOVE SPACES       TO MSG-LOCKWORD.
     MOVE 'D'          TO MSG-OPTION.
     MOVE '5PD'        TO MSG-TRXID.
     MOVE '02AN960C10' TO MSG-KEY.
*    NOTE ***  SET ALT  PCB DESTINATION TO SIGNON TRANSACTION.
     CALL 'CBLTDLI' USING CHNG ALT-PCB1 SPATRANS.
     IF NOT ALT-OK1 MOVE ALT-SYS TO DISP-STAT GO TO DISPLY-STAT.
*    NOTE ***  INSERT SPA TO TRANSACTION REQUESTED BY AUDIT EXIT.
     CALL 'CBLTDLI' USING ISRT ALT-PCB1 SPADSECT.
     IF NOT ALT-OK1 MOVE ALT-SYS TO DISP-STAT GO TO DISPLY-STAT.
*    NOTE ***  INSERT MSG TO TRANSACTION REQUESTED BY AUDIT EXIT.
     CALL 'CBLTDLI' USING ISRT ALT-PCB1 MSG.
     IF NOT ALT-OK1 MOVE ALT-SYS TO DISP-STAT GO TO DISPLY-STAT.
*    NOTE ***  RETURN TO ADF.
     GO TO RE-EXIT.
DISPLY-STAT.
*    NOTE ***  THIS PARAGRAPH DISPLAYS DL/1 STATUS INFO
*              IN THE EVENT OF AN ERROR.
     DISPLAY 'TERM STATUS = ' LT-STATUS 'DB-STATUS = '
     ADFW-STATUS 'PROG-STATUS = ' DISP-STAT.
RE-EXIT.
    GOBACK.
```

The link-edit control statements required are:

```
  ENTRY DLITCBL
  NAME SAMPTRS(R)
```

SWITCHING DIRECTLY TO THE DRIVER

This is the more efficient of the two techniques. It has some disadvantages:

- The COBOL program must set flags depending on whether the target IMSADF II transaction requires key selection, special processing or text utility; this is information that IMSADF II derives from the Secondary Option Menu Rule.

- It can only be used when the COBOL program itself was invoked by a switch from IMSADF II. This is because IMSADF II must format the SPA or the IMSADF II Work Data Base correctly.

The COBOL program (MPP) receives the SPA as formatted by IMSADF II and determines from the length whether it is a full SPA or short SPA (28 bytes) requiring retrieval from the Work Data Base. The program can use and alter data in the communication area (SPAFLDSG).

In order to switch back to IMSADF II, it must perform the following flag setting in the SPADSECT:

- Set the new IMS/VS transaction code in SPATRANS and SPASHOTR.

- Set the new IMSADF II TRXID in SPACGTRX and SPATRX.

- Move binary zero into SPAFIRST, SPARTNCD, SPASECTX, SPAPGOPT, SPABITS, and SPASWITH.

- Set SPASEGUT to binary zero and then set three bits within it, depending on whether the next IMSADF II transaction:

  - requires key selection (SPASPUTL)
  - uses special processing (SPASEGUT)
  - uses text utility   (SPATXTUT)

  In COBOL, these bits are set by adding appropriate powers of 2 (as shown in the sample program below).

Finally, the program must issue the IMS/VS CHNG and ISRT calls to complete the switch, as documented in the IMS/VS Application Programming Manual.

The following program is intended as an illustration of how to use the technique. (It assumes that IMSADF II sign-on has been performed to set up the SPA and/or the Work Data Base.)

```
IDENTIFICATION DIVISION.
PROGRAM-ID.
    SAMPTSW.
AUTHOR.
    IMSADF II.
DATE-WRITTEN.
    1982.
REMARKS.
        THIS SKELETON PROGRAM RECEIVES CONTROL FROM AN IMSADF II
    TRANSACTION VIA AN IMS/VS CONVERSATIONAL PROGRAM TO PROGRAM
    SWITCH AND READS THE SPA AND, IF NECESSARY, THE ROOT SEGMENT
    OF THE IMSADF II WORK DATA BASE.  CODE COULD BE INSERTED AT
    THIS POINT TO PERFORM THE DESIRED APPLICATION PROGRAM
    FUNCTIONS.  FOLLOWING THE APPLICATION CODE, THIS SKELETON
    PROGRAM SETS THE NECESSARY SWITCHES AND TRANSACTION CODES
    IN THE SPA AND WORK DATA BASE, THEN UPDATES THE WORK DATA
    BASE AND INSERTS THE SPA TO THE MESSAGE QUEUE.  IMS/VS WILL
    THEN SCHEDULE THE NEXT IMSADF II TRANSACTION.
ENVIRONMENT DIVISION.
CONFIGURATION SECTION.
SOURCE-COMPUTER.
    IBM-370.
OBJECT-COMPUTER.
    IBM-370.
INPUT-OUTPUT SECTION.
FILE-CONTROL.
DATA DIVISION.
FILE SECTION.
WORKING-STORAGE SECTION.
77  GU PIC XXXX VALUE 'GU  '.
77  ISRT PIC XXXX VALUE 'ISRT'.
77  REPL PIC XXXX VALUE 'REPL'.
77  CHNG  PIC XXXX VALUE 'CHNG'.
77  GHU PIC XXXX VALUE 'GHU '.
77  DISP-STAT PIC X(12) VALUE SPACES.
77  LT-SYS PIC X(12) VALUE 'LT-SYS-ERR.'.
77  DA-SYS PIC X(12) VALUE 'DA-SYS-ERR.'.
77  ALT-SYS PIC X(12) VALUE 'ALT-SYS-ERR.'.
*    THE FOLLOWING FLAGS ARE NEEDED FOR REQUESTING KEY SELECTION,
*    SPECIAL PROCESSING AND TEXT UTILITY.
77  SPASEGUT-I PIC 9(4) COMP VALUE 2048.
77  SPASPUTL-I PIC 9(4) COMP VALUE 256.
77  SPATXTUT-I PIC 9(4) COMP VALUE 32.
*    SEGMENT SEARCH ARGUMENTS FOR WORK DATA BASE.
01  SSA1.
    02 SSA1-SEG PIC X(19) VALUE 'ADFWORK1(LTERMNME ='.
    02 SSA1-KEY PIC X(8).
    02 SSA1-QUAL PIC X VALUE ')'.
01  SSA2.
    02 SSA2-SEG PIC X(8) VALUE 'ADFWORK1'.
    02 FILLER   PIC XX   VALUE '  '.
*    SAVE AREA FOR SMALL IMS SPA.
01  IMS-SPA         PIC X(28) VALUE SPACES.
*    IF WORK DATA BASE USED, MUST ALLOW FOR PRECEDING 8 BYTE KEY.
01  ADFWK-AREA.
  03  ADFWK-KEY     PIC X(8).
COPY SPACOBOL.
    05  MYAREA REDEFINES SPAFLDSG PIC X.
*      NOTE ***  IF COMMLEN IS DEFINED, THE USER DATA FROM OR TO
*                AN AUDIT EXIT WILL BE IN SPAFLDSG.
  03  FILLER  PIC X(29967).
01  SPA-START REDEFINES SPADSECT PIC X(28).
```

```
LINKAGE SECTION.
01  LT-PCB.
    02  LTERM  PIC X(8).
    02  FILLER PIC XX.
    02 LT-STATUS PIC XX.
88  Q-EMPTY VALUE 'QC'.
88  LT-OK VALUE '  '.
    02 FILLER PIC X(12).
01  ALT-PCB1.
    02  FILLER PIC X(10).
    02 ALT-STAT1 PIC XX.
88  AQ-EMPTY1 VALUE 'QC'.
88  ALT-OK1 VALUE '  '.
    02 FILLER PIC X(12).
01  ALT-PCB2.
    02  FILLER PIC X(10).
    02 ALT-STAT2 PIC XX.
88  AQ-EMPTY2 VALUE 'QC'.
88  ALT-OK2 VALUE '  '.
    02 FILLER PIC X(12).
01  ADFWK-PCB.
    02 FILLER PIC X(10).
    02  ADFW-STATUS PIC XX.
88  ADF-NOTFND VALUE 'GE'.
88  ADFW-OK VALUE '  '.
PROCEDURE DIVISION.
    ENTRY 'DLITCBL' USING LT-PCB, ALT-PCB1, ALT-PCB2, ADFWK-PCB.
RE-START.
*    NOTE ***  RETRIEVE SPA FROM IMS/VS.
     CALL 'CBLTDLI' USING GU LT-PCB SPADSECT.
     IF Q-EMPTY GO TO RE-EXIT.
     IF NOT LT-OK MOVE LT-SYS TO DISP-STAT GO TO DISPLY-STAT.
*    NOTE ***  IS SMALL SPA FOUND,  RETRIEVE FROM WORK DATA BASE.
     IF SPALEGTH = 28 PERFORM READ-ADFWRK THRU END-READ-ADFWRK.
*            APPLICATION CODING STARTS HERE.
     DISPLAY 'SWITCH PROGRAM EXECUTING'.
*            APPLICATION CODING FINISHES HERE.
PREP-SWITCH.
*    NOTE ***  NOW PREPARE SPA AND WORK D-B FOR RETURN TO IMSADF II
     MOVE ZERO TO SPAFIRST SPARTNCD SPASECTX SPAPGOPT.
     MOVE LOW-VALUES TO SPABITS SPASWITH.
*    NOTE ***  IN THIS EXAMPLE, WE SWITCH TO THE CD TRXID.
     MOVE 'SAMPVCD ' TO SPATRANS SPASHOTR.
     MOVE '5CD' TO SPACGTRX SPATRX.
*    NOTE ***  TO SET BITS ON IN COBOL, ADD THE OPTIONS.
*              THESE OPTIONS TELL THE DRIVER WHETHER THE NEXT TRX
*              (1) REQUIRES KEY SELECTION (SPASPUTL-I)
*              (2) USES SPECIAL PROCESSING (SPASEGUT-I)
*              (3) USES TEXT UTILITY (SPATXTUT-I).
*    NOTE ***  THE CD TRANSACTION USES KEY SELECTION AND
*              SPECIAL PROCESSING.
     COMPUTE SPASWITH-R = SPASPUTL-I + SPASEGUT-I.
*    NOTE ***  IF SMALL SPA IN USE, REPLACE WORK DATA BASE SEG.
     IF IMS-SPA NOT = SPACES
       PERFORM REPL-ADFWRK THRU END-REPL-ADFWRK.
*    NOTE ***  SET ALT PCB DESTINATION TO NEW TRANSACTION.
     CALL 'CBLTDLI' USING CHNG ALT-PCB1 SPATRANS.
     IF NOT ALT-OK1 MOVE ALT-SYS TO DISP-STAT GO TO DISPLY-STAT.
*    NOTE ***  INSERT SPA TO NEW TRANSACTION
     CALL 'CBLTDLI' USING ISRT ALT-PCB1 SPADSECT.
     IF NOT ALT-OK1 MOVE ALT-SYS TO DISP-STAT GO TO DISPLY-STAT.
*    NOTE ***  RETURN TO ADF.
     GO TO RE-EXIT.
DISPLY-STAT.
*    NOTE ***  THIS PARAGRAPH DISPLAYS DL/1 STATUS INFO
*              IN THE EVENT OF AN ERROR.
     DISPLAY 'TERM STATUS = ' LT-STATUS 'DB-STATUS = '
     ADFW-STATUS 'PROG-STATUS = ' DISP-STAT.
RE-EXIT.
    GOBACK.
```

```
READ-ADFWRK.
*    NOTE ***  COPY 28 BYTE SPA INTO SEPARATE AREA
*              BEFORE READING FROM WORK DATA BASE.
     MOVE SPA-START TO IMS-SPA.
*    NOTE ***  USE IOPCB LTERM NAME AS KEY FOR WORK D-B.
     MOVE LTERM TO SSA1-KEY.
*    NOTE ***  RETRIEVE ROOT SEGMENT OF WORK D-B.
     CALL 'CBLTDLI' USING GHU ADFWK-PCB ADFWK-AREA SSA1.
     IF NOT ADFW-OK MOVE DA-SYS TO DISP-STAT GO TO DISPLY-STAT.
END-READ-ADFWRK.
REPL-ADFWRK.
*    NOTE ***  NOW REPLACE WORK D-B ROOT SEGMENT.
     CALL 'CBLTDLI' USING REPL ADFWK-PCB ADFWK-AREA SSA2.
     IF NOT ADFW-OK MOVE ALT-SYS TO DISP-STAT GO TO DISPLY-STAT.
*    NOTE ***  COPY 28 BYTE SPA BACK FROM SEPARATE AREA.
     MOVE IMS-SPA TO SPA-START.
END-REPL-ADFWRK.
```

The link-edit control statements required are:

```
  ENTRY DLITCBL
  NAME SAMPTSW(R)
```

INDEX

IMS Application Development Facility II Version 2 Release 2
Application Development Guide
SH20-6595-01

READER'S COMMENT FORM

You may use this form to communicate your comments about this publication, its organization, or subject matter, with the understanding that IBM may use or distribute whatever information you supply in any way it believes appropriate without incurring any obligation to you.

Your comments will be sent to the author's department for whatever review and action, if any, are deemed appropriate.

Note: *Copies of IBM publications are not stocked at the location to which this form is addressed. Please direct any requests for copies of publications, or for assistance in using your IBM system, to your IBM representative or to the IBM branch office serving your locality.*

Possible topics for comment are:

Clarity Accuracy Completeness Organization Coding Retrieval Legibility

If you wish a reply, give your name, company, mailing address, and date:

____

____

____

____

What is your occupation? ______________________________

Number of latest Newsletter associated with this publication: ____________________

Thank you for your cooperation. No postage stamp necessary if mailed in the U.S.A. (Elsewhere, an IBM office or representative will be happy to forward your comments or you may mail directly to the address in the Edition Notice on the back of the title page.)

SH20-6595-01

Reader's Comment Form

Cut or Fold Along Line

Fold and tape | Please Do Not Staple | Fold and tape

NO POSTAGE
NECESSARY
IF MAILED
IN THE
UNITED STATES

BUSINESS REPLY MAIL

FIRST CLASS PERMIT NO. 40 ARMONK, N.Y.

POSTAGE WILL BE PAID BY ADDRESSEE:

International Business Machines Corporation
Department 8D8
220 Las Colinas Boulevard
Irving, Texas 75039-5513

Fold and tape | Please Do Not Staple | Fold and tape

IBM®

IMS Application Development Facility II Version 2 Release 2
Application Development Guide
SH20-6595-01

READER'S COMMENT FORM

You may use this form to communicate your comments about this publication, its organization, or subject matter, with the understanding that IBM may use or distribute whatever information you supply in any way it believes appropriate without incurring any obligation to you.

Your comments will be sent to the author's department for whatever review and action, if any, are deemed appropriate.

Note: *Copies of IBM publications are not stocked at the location to which this form is addressed. Please direct any requests for copies of publications, or for assistance in using your IBM system, to your IBM representative or to the IBM branch office serving your locality.*

Possible topics for comment are:

Clarity Accuracy Completeness Organization Coding Retrieval Legibility

If you wish a reply, give your name, company, mailing address, and date:

____

____

____

____

What is your occupation? ______________________________

Number of latest Newsletter associated with this publication: ____________________

Thank you for your cooperation. No postage stamp necessary if mailed in the U.S.A. (Elsewhere, an IBM office or representative will be happy to forward your comments or you may mail directly to the address in the Edition Notice on the back of the title page.)

SH20-6595-01

Reader's Comment Form

Cut or Fold Along Line

Fold and tape | Please Do Not Staple | Fold and tape

NO POSTAGE
NECESSARY
IF MAILED
IN THE
UNITED STATES

BUSINESS REPLY MAIL

FIRST CLASS PERMIT NO. 40 ARMONK, N.Y.

POSTAGE WILL BE PAID BY ADDRESSEE:

International Business Machines Corporation
Department 8D8
220 Las Colinas Boulevard
Irving, Texas 75039-5513

Fold and tape | Please Do Not Staple | Fold and tape

IBM®

IMS Application Development Facility II Version 2 Release 2
Application Development Guide
SH20-6595-01

READER'S COMMENT FORM

You may use this form to communicate your comments about this publication, its organization, or subject matter, with the understanding that IBM may use or distribute whatever information you supply in any way it believes appropriate without incurring any obligation to you.

Your comments will be sent to the author's department for whatever review and action, if any, are deemed appropriate.

Note: *Copies of IBM publications are not stocked at the location to which this form is addressed. Please direct any requests for copies of publications, or for assistance in using your IBM system, to your IBM representative or to the IBM branch office serving your locality.*

Possible topics for comment are:

Clarity Accuracy Completeness Organization Coding Retrieval Legibility

If you wish a reply, give your name, company, mailing address, and date:

____

____

____

____

What is your occupation? ____

Number of latest Newsletter associated with this publication: ________________________

Thank you for your cooperation. No postage stamp necessary if mailed in the U.S.A. (Elsewhere, an IBM office or representative will be happy to forward your comments or you may mail directly to the address in the Edition Notice on the back of the title page.)

SH20-6595-01

Reader's Comment Form

Cut or Fold Along Line

Fold and tape

Please Do Not Staple

Fold and tape

NO POSTAGE
NECESSARY
IF MAILED
IN THE
UNITED STATES

International Business Machines Corporation
Department 8D8
220 Las Colinas Boulevard
Irving, Texas 75039-5513

Fold and tape

Please Do Not Staple

Fold and tape

IBM ®

IMS Application Development Facility II Version 2 Release 2
Application Development Guide
SH20-6595-01

READER'S COMMENT FORM

You may use this form to communicate your comments about this publication, its organization, or subject matter, with the understanding that IBM may use or distribute whatever information you supply in any way it believes appropriate without incurring any obligation to you.

Your comments will be sent to the author's department for whatever review and action, if any, are deemed appropriate.

Note: *Copies of IBM publications are not stocked at the location to which this form is addressed. Please direct any requests for copies of publications, or for assistance in using your IBM system, to your IBM representative or to the IBM branch office serving your locality.*

Possible topics for comment are:

Clarity Accuracy Completeness Organization Coding Retrieval Legibility

If you wish a reply, give your name, company, mailing address, and date:

____

____

____

____

What is your occupation? ____

Number of latest Newsletter associated with this publication: ____________________

Thank you for your cooperation. No postage stamp necessary if mailed in the U.S.A. (Elsewhere, an IBM office or representative will be happy to forward your comments or you may mail directly to the address in the Edition Notice on the back of the title page.)

SH20-6595-01

Reader's Comment Form

Cut or Fold Along Line

Fold and tape Please Do Not Staple Fold and tape

International Business Machines Corporation
Department 8D8
220 Las Colinas Boulevard
Irving, Texas 75039-5513

Fold and tape Please Do Not Staple Fold and tape

IBM®

IBM

Printed in U.S.A.