Publication Number
SH20-6594-01

IMS Application Development Facility II
Version 2 Release 2

Application Development Reference

IBM®

IMS Application Development Facility II
Version 2 Release 2

Application Development Reference

Publication Number
SH20-6594-01

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 provides reference data for developing IMSADF II applications.

The manual consists of eight chapters and six appendixes.

- **Chapter 1, "System Overview"** describes the development of an IMSADF II application in general terms.
- **Chapter 2, "Rules Generator"** contains detailed information about the generation of static rules and screens.
- **Chapter 3, "Audit Language"** provides detailed information on the language used to define IMSADF II audits.
- **Chapter 4, "Dynamic Rules Data Bases"** provides detailed information about the IMSADF II Dynamic Rules Data Bases.
- **Chapter 5, "Audit Logic Processing"** contains detailed information about the IMSADF II Audit Data Base and describes the operation of the IMSADF II Auditor.
- **Chapter 6, "Special Processing and User Exits"** describes how to implement user written code.
- **Chapter 7, "IMSADF II under IMS/VS"** explains the procedures for defining an IMSADF II application to IMS/VS.
- **Chapter 8, "IMSADF II under CICS/OS/VS"** explains the procedures for defining an IMSADF II application to CICS/OS/VS.
- The appendixes include:

  Appendix A, "Summary of Rules Generator Operand Usage"

  Appendix B, "Data Conversions and Mappings"

  Appendix C, "Naming Conventions"

  Appendix D, "Additional Application Support Techniques"

  Appendix E, "Error Messages - Rules Generator"

  Appendix F, "Error Messages - Audit Language"

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 Guide, 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. SYSTEM OVERVIEW

User specified rules that define IMSADF II applications are stored as load modules (members of an OS partitioned data set) and data base records (DL/I or DB2). IMSADF II rules can contain definition of:

- systems access (controlled by user ID, project code, etc.)
- type of access (update or retrieve only)
- DL/I data base structure and segment layouts
- DB2 tables and row layouts
- audit criteria (for example, update is permitted only if input falls within a specified range)
- message transmission (notify certain system users if particular records have been altered)

The development of an application under IMSADF II requires a variety of design considerations and implementation steps. Following the data base design, a number of steps are required to implement the application. These steps are detailed for a sample application using Standard Processing in the Sample Problem in the IMS Application Development Facility II Version 2 Release 2 Installation Guide.

By implementing application variables in terms of rules stored externally to the program, a new level of processing flexibility and data independence is achieved. The tasks of designing, implementing, and modifying applications and their data bases and of integrating these applications and data bases will be faster and more precise.

RULES

IMSADF II rules are two types, Static and Dynamic.

STATIC RULES

Static rules are created through input specifications to the IMSADF II Rules Generator and reside as load modules in an Operating System Partitioned Data Set.

- **Primary Option Menu Rule**

  The Primary Option Menu Rule (POMR) controls those major functions that are allowed in a particular application system such as message sending and display, and Transaction selection. The application designer builds a Primary Option Menu Rule for each application system. An application system can be defined as a set of related applications for one or more Project/Groups.

- **Secondary Option Menu Rule**

  The Secondary Option Menu Rule (SOMR) defines all possible IMSADF II Transactions in an application system. Using this rule and the Sign-on Profile Data Base, IMSADF II presents a menu screen that displays only those transactions that are authorized for that particular end user.

- **Input Transaction Rule**

  The Input Transaction Rule (ITR) defines the transaction type (conversational, nonconversational, or batch), the segments to be retrieved, and the fields to be displayed and audited. This rule indicates whether the processing to be performed will be Standard Processing or if a Special Processing Routine needs to be called to perform specific application logic. In addition, the Input

Transaction Rule can indicate that a secondary transaction should be sent to another application program or to a logical terminal.

- **Output Format Rule**

  The Output Format Rule (OFR) is related to the Input Transaction Rule in that it defines the content of a message that is sent either to a logical terminal for display, to another application program for further processing or to an OS/VS sequential data set. The format of this rule is similar to that of the Segment Layout Rule, which is discussed below.

- **Segment Layout Rule**

  The Segment Layout Rule (SLR) defines contents of a data base segment, pseudo segment, or data area. Standard IMSADF II access to the segment or area is controlled by the Segment Layout Rule without requiring user code.

  Each Segment Layout Rule defines each field in a physical or logical data base segment, the location of each field in a segment, and the type of data representation within a segment. A Segment Layout Rule controls whether or not a changed field in a data base segment is to be audited, and indicates automatic message sending requirements.

  A pseudo segment is a work area that can be used for developing intermediate results for reference and display.

  A Segment Layout Rule is also used to describe a data area that has been defined in a Special Processing program. This version of the Segment Layout Rule will be referred to as a mapping rule. With the use of the data mapping technique, data is passed to the Special Processing program in the location and form requested. In this way, Special Processing programs achieve the same degree of data independence as is available with Standard Processing.

- **Segment Handler Rule**

  The Segment Handler Rule (SHR) defines the communication between IMSADF II modules and data bases. All I/O for a particular segment is provided by the Segment Handler module by interpreting the Segment Handler Rule. A Segment Handler Rule is created by the Rules Generator for each data base segment type to be accessed by the Common Segment Handler module.

- **Table Handler Rule (DB2 Only)**

  The Table Handler Rule (THR) contains the compiled SQL statements to retrieve and update a DB2 table. The rule contains standard IMSADF II statements to select, update, insert, delete, and browse. In addition, user defined statements may be included to provide unique selection criteria. This rule is used when any I/O is performed against a DB2 table.

- **Table Layout Rule (DB2 Only)**

  This rule is much like a Segment Layout Rule except it defines the layout of a DB2 row. It defines location and attributes of columns and whether NULL values are valid.

DYNAMIC RULES

Dynamic rules are stored as segments in data bases (DL/I or DB2), and can be created and modified online using facilities provided by IMSADF II.

- **Audit Rule**

  This rule defines the type of audit/logic routines to be performed on a field. In addition to field audits, this rule controls automatic field assignment (assignment of values to a field depending on pre-set values or contents of related fields) and message sending (automatic notification when a field has been

changed). The rule is comprised of segments in the Audit Data Base or optionally may be created in load module format.

- **Sign-on Profile Rule**

  The Sign-on Profile Rule specifies security information for user access to the system. In addition, transaction sensitivity can be controlled by specifying transaction IDs and modes for which a user is authorized. This rule resides in the Sign-on Profile Data Base.

- **Message Rule**

  The Message Rule controls message generation, informational message routing, secondary transaction destination and user ID mailboxes. Messages for error, warning or informational situations are contained in the Message Data Base. The developer has the option of specifying additional HELP text to be associated with each message. This HELP text is also contained in the Message Data Base. For further information on this refer to "HELP Facility" on page 4-13 in Chapter 4, "Dynamic Rules Data Bases" of this manual. Routing information for informational messages and secondary transactions is also contained in the Message Data Base to direct messages to projects, user IDs, logical terminals or transactions. Additionally, the Message Data Base provides a collection facility for messages sent by other users or messages sent automatically by IMSADF II as a result of data base updates.

PSEUDO SEGMENTS

The Input Transaction Rule describes three types of segments that are used in processing. These segment types include data base segments loaded by the transaction driver, data base segments loaded by a Special Processing routine or Auditor routine, and pseudo segments.

Data base segments loaded by the transaction driver are specified by the DBPATH Rules Generator Language operand and may come from multiple hierarchical paths with multiple data bases. Data base segments not loaded by the transaction driver are specified by the TSEGS Rules Generator Language operand. Space is reserved for subsequent loads and updates by the Special Processing routine.

Pseudo segments contain fields that are not present in data base segments. For example, you can use pseudo segments to contain data entered from the terminal that does not map directly into a data base segment field. A pseudo segment is treated the same as a data base segment which is not loaded. Space is reserved in the SPA segment area for the pseudo segment and the Segment Layout Rule that describes it is loaded. Subsequent input data either is mapped automatically from transaction input or can be mapped by the Special Processing Routine into these pseudo segments fields through a call to the Service Routine Data Mapper.

Pseudo segments can be used in either Standard or Special Processing. Pseudo segments are specified by the TSEGS Rules Generator Language operand.

NAMING CONVENTIONS

The IMSADF II architecture dictates the use of several naming conventions unique to IMSADF II application rules. These unique naming conventions apply to:

- Major Application System ID - a four-character code used to identify the components of an IMSADF II application system. The first two characters should be unique with regard to other application systems using IMSADF II techniques. The first of these two characters must be alphabetic. If the first two characters are not unique, then a unique MFS trailer code must be specified for each major system ID.

- Project/Group Code - a two-character code that identifies the role or job class applicable to a set of users. This code is the key of the root segment for the Sign-on Profile Data Base. The code reflects the organizational hierarchy required to effect access control. Individual end users can be identified uniquely within a Project/Group category by the use of serial numbers. Each specific authority profile is related to an individual user within one or more Project/Groups. The authority profile for an individual may be different within different Project/Groups. This code also is used to uniquely identify the Input Transaction Rule for a given session function.

- Segment ID - a two-character code used to identify Facility segment definitions. Segment IDs must be unique within a Major Application System ID. These definitions may apply to data base, pseudo, or mapping segments. This convention does not dictate installation standards in regard to data base and segment names within IMS/VS Data Base Descriptions (DBDs).

- Transaction ID - a two-character code used to identify a Standard or Special Processing transaction. Transaction IDs must be unique within a Major Application System ID.

Although Segment IDs and Transaction IDs must each be unique within a Major Application System ID, they may overlap each other since they are used in the naming of different rules.

DEVELOPMENT OF A STANDARD PROCESSING APPLICATION

Since the majority of applications developed under IMSADF II will use Conversational Standard Processing, the following section describes the steps to develop a Conversational Standard Processing application.

Assume for this example the data base illustrated in Figure 1-1.

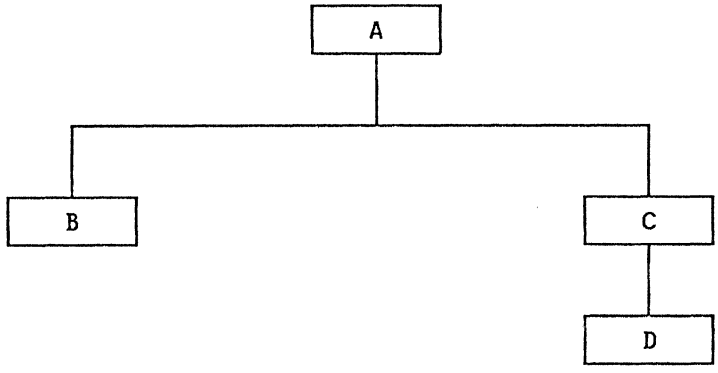


Figure 1-1. Example Data Base

This data base consists of four segment types and is three levels deep. Assume that the application must:

1. Allow display and update of segments A, C, and D.
2. Provide for field auditing of all fields updated.
3. Provide for automatic message sending if specific fields are updated in the segments.
4. Provide assistance to the terminal user in locating segment type D.

- PRELIMINARY ACTIVITIES

  1. Specify a four-character system ID. The first two characters are unique to the application.
  2. Specify two-character segment IDs for segments A, C, and D. These characters are unique to the segment within this application.
  3. Specify two-character transaction ID used to identify the processing function (also used name the Input Transaction Rule).
  4. Specify two-character project/group IDs. These characters specify the different project/groups that can access the transactions.

- DATA BASE ACTIVITIES

  1. Review the data base design.
  2. Generate the IMS/VS DBDs.
  3. Generate or update the IMS/VS PSBs for Transaction Driver transactions.
  4. Generate the IMS/VS ACBs.

- RULE ACTIVITIES

  The following rules are created using the facilities of the Rules Generator and IMSADF II dynamic rules data base update transactions.

  1. Sign-on Profile Rule - Segments are added to the Sign-on Profile Data Base to reflect:

     - Project/Groups that can sign on.

     - USERIDs that are allowed under each Project Group.

     - Transaction IDs to which each user has access and the transaction modes (add, update, retrieve, etc.) that each user can use.

  2. Primary Option Menu Rule - Each application System ID has a Primary Option Menu Rule that specifies the options available to the major application system. In most case, the Primary Option Menu Rule will be the same for each application.

  3. Secondary Option Menu Rule - A Secondary Option Menu Rule will be created to specify the Transaction ID that can be accessed with this application.

  4. Input Transaction Rule - The transaction driver uses an Input Transaction Rule to determine which segment or segments should be loaded into the SPA for display and/or update. In the example, segments A, C, and D should be loaded and displayed on one screen. Specific fields from each segment will be displayed at predetermined screen locations. The Input Transaction Rule will specify which fields in the segments should be displayed and at which screen location. One Input Transaction Rule will be required for this example.

  5. Segment Layout Rule - One Segment Layout Rule is created for each segment type accessed. This rule specifies the attributes and position of each field in a segment. It also indicates information concerning audit requirements. One Segment Layout Rule each should be generated for segment types A, C, and D. Each of these rules should specify that all the fields which are displayed should be audited if modified.

  6. Audit Rule - Segments should be included in the Audit Data Base to control the audit routines that will be in effect for modified fields. Each field will have a unique set of Audit Rules. All displayed fields for segment types A, C, and D will have Audit Rules specified. In addition, if notification is to be sent to another destination upon a valid update of a field, the message sending requirement also must be specified in these rules.

  7. Message Rule - This rule describes the destination or destinations to which a field update message is to be sent. It also indicates the message text. A separate rule is required for each field update that requires message sending.

  8. Segment Handler Rule - This rule describes the IMS/VS segment search arguments and segment key fields needed to retrieve a segment. A Segment Handler Rule is needed for each segment type (A, C, and D) that is read.

- HOST SYSTEM ACTIVITIES

  1. Screen source is created by the Rules Generator to be processed by the host system utility:

     - IMS/VS - Message Format Services Utility

     - CICS/OS/VS - Basic Mapping Support Utility

     Screen source is created by the Rules Generator for:

- Sign-on Screen - for each major application system identification

- Primary Key Selection Screen - used in conjunction with the Input Transaction Rule in the key selection process.

- Segment Display Screen - describes the layout of the message segments displayed by the transaction driver.

2. The host system, IMS/VS or CICS/OS/VS, generally requires some definition of IMSADF II components in their system definition. For IMS/VS, the APPLCTN and TRANSACT macros define the application programs and transactions codes. For CICS/OS/VS, DFHPPT and DFHPCT macros define the application programs and transaction codes.

The foregoing example was designed to familiarize the application designer with the various steps needed to install an application. Although several steps are required, all rules and screens can be generated with a single pass through the Rules Generator. In addition, any future modifications to segment layout, screen layout, or audit requirements can be made with a minimum of effort. Refer to the IMS Application Development Facility II Version 2 Release 2 Application Development Guide for an example of input to the Rules Generator for the Sample Problem.

RECOMMENDED PROCEDURE FOR MANAGING THE CREATION OF RULES AND SCREENS FOR TRANSACTIONS IN THE SAME APPLICATION SYSTEM ID.

Since generated rules and screens may be shared by all transactions as well as be replaced by newly generated rules and screens with the same names within the same application system ID, the following procedure for controlling the generation of rules is recommended.

- The generation of all segment related rules for data base segments should be controlled by one person such as a data base administrator. These rules should be generated for all segments in a data base and need to be generated only once, unless subsequent modifications are made to the data base segment definitions. These include the Segment Layout Rules, which basically define a structure for the data base segments and the Segment Handler Rules which contain the segment search arguments for IMS/VS. These rules may then be used by all transactions that access related data base segments in the same application system ID.

- The developer/programmer need only be concerned with the transaction related rules and screen formats. These include the Input Transaction Rules, which define what the transaction does, the Segment Display Screen, and the Key Display Screen. If mapping or pseudo segments are used, then the developer must also generate Segment Layout Rules for these segment types.

- Application system related rules and screens including the Sign-on Screen, Primary Option Menu Rule (POMR) and Secondary Option Menu Rule (SOMR), should be handled as follows. The first conversational transaction within an application system must create a Sign-on Screen and a Primary Option Menu Rule through using the OPTIONS=CVSYS. These rules need to be created only once using the application system ID. The Secondary Option Menu Rule contains all transaction IDs within an application system. For every Conversational (regular or text) Input Transaction Rule created, an entry needs to be added to SOMR, using OPTION=SOM in the final GENERATE statement of a run. Application system related rules and screens apply only to Conversational processing.

- Finally, a transaction driver load module must be created through the link-edit options of the generate statement. Although one transaction driver could conceivably process all transactions within an application system, there is typically more than one.

DL/I DATA BASE DESIGN CONSIDERATIONS

The design of DL/I data bases can affect the facilities that are available to the application. Standard Processing offers the highest degree of data independence and maintainability. In order to use this technique, data bases should be designed so that data which need to be displayed and updated as a unit fall within one or more hierarchical path(s). Each segment type that is supported in a data base must contain at least one DBD sequence field. Standard Processing supports single occurrences of a segment type at multiple levels within one or more hierarchical path(s). Up to one hundred segment occurrences within the paths can be displayed for update in a single iteration. The fully concatenated keys of the segment(s) retrieved from the hierarchical path(s) are constructed either from transaction input (screen or batch) or fields of previously loaded segments. Standard Processing also supports:

1. A fully concatenated key up to 255 bytes in length for a single path and 255 bytes for all paths.

2. Key of an individual segment up to 70 bytes in length.

3. Segments up to 6000 bytes in length.

4. Mixed mode access to Fastpath data bases.

If a data base cannot be designed to meet the areas supported, a Special Processing Routine can be written to handle specific DL/I input/output requirements.

CHAPTER 2. RULES GENERATOR

INTRODUCTION

The Rules Generator is an executable module that is used to develop static rules and screen source statements for implementing application systems. The Rules Generator is invoked by the ????G JCL procedure (???? are changed to the IMSADF II System ID at installation) and run under OS/VS as a single job step. The input to the Rules Generator is standard OS/VS sequential 80-byte record input. The output from the Rules Generator consists of:

- transaction drivers
- static rules
- screen source (IMS/VS MFS or CICS/OS/VS BMS)

The transaction driver modules are placed in a program load library. This library name can be changed through Rules Generator parameters when these modules are link-edited.

The Rules Generator automatically generates rule, segment, field, and screen names according to IMSADF II conventions. It can generate rule object decks directly or by invoking the Assembler. When all input data has been processed and no errors have been encountered, the Rules Generator invokes the Linkage Editor to store each rule in load module format as a member of a specified rules load library.

The IMS/VS Message Format Services (MFS) screen source output is written to the sequential data set specified with DDNAME=SCREENS. All the screen formats for one major application system can be built in a single MFS run.

The CICS/OS/VS Basic Mapping Support (BMS) screen source output is written to an OS partitioned data set (PDS) specified with DDNAME=CICSMAPS. Each member of the PDS is a **mapset**, containing the maps for one IMSADF II transaction.

transaction drivers:

- Conversational (Standard and Special Processing)
- Nonconversational (Standard and Special Processing)
- Batch (Standard and Special Processing)

static rules:

- Primary Option Menu Rule for Conversational Processing
- Secondary Option Menu Rule for Conversational Processing
- Conversational Input Transaction Rule
- Nonconversational Input Transaction Rule
- Batch Input Transaction Rule
- Segment Layout Rule for data base segments
- Segment Layout Rule for pseudo segments
- Segment Layout Rule for mapping segments (Mapping Rule)
- Segment Layout Rule for output segments (Output Format Rule)
- Segment Handler Rule
- Conversational Input Transaction Rule for text segments
- Batch Driver Rule (included in batch transaction driver executable load module)
- Preload Rule

screen source:

- Sign-on screen
- Primary Key Selection screen
- Segment Display screens for Conversational and Nonconversational processing
- Text Utility Screen
- Secondary Transaction Output

RULES GENERATOR CONTROL STATEMENT OVERVIEW

The Rules Generator language consists of six control statements:

- SYSTEM
- SEGMENT/TABLE
- FIELD/COLUMN
- GENERATE
- RULE
- INCLUDE

The SYSTEM statement provides control information pertaining to the Rules Generator execution. Such information as System ID, Project/Group and screen heading appear on this statement. Some operands on the SYSTEM statement can be changed on subsequent SEGMENT and GENERATE statements. More than one SYSTEM statement can appear in the input stream. Each occurrence of a SYSTEM statement initializes the rules generation process.

The SEGMENT and FIELD statements are used to define:

- IMSADF II's view of a data base segment
- a pseudo segment
- a mapping segment
- an output segment

These segments will be referred to as "application segments". Each application segment definition includes a SEGMENT statement and one or more FIELD statements immediately following the SEGMENT statement. They contain information about, processing requirements for, and the fields stored within data base, pseudo, mapping, and output segments. Only those hierarchical paths that require processing by IMSADF II need to be specified to the Rules Generator. FIELD statements may be specified in either keyword format or tabular format. In the latter case, the keywords are not entered, and each operand value must be entered within fixed column locations.

The GENERATE statement is used to request rules and screen generation. If segment definitions are entered without generation requests or generation requests for segment or transaction rules are entered without segment definitions, no generation will occur and no error messages will be produced to indicate that rules or screens have not been generated.

The RULE statement is provided for invoking the Rules Generator assembly and link-edit phases. This feature is only used if a deviation from the standard Rules Generator conventions is required. The RULE statement is automatically added to the Assembler Language rule source decks created when a deck request is specified. After the Assembler source has been modified, the Rules Generator may be reentered using the existing RULE statement.

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

The INCLUDE statement allows insertion of previously defined common definitions into the Rules Generator input stream. The definition may consist of Rules Generator Control statements and screen image layout parameters.

Figure 2-1 shows the hierarchy of the Rules Generator control statements.

```
SYSTEM       Provide identification characters for major application
             system, and project group:

 SEGMENT     Provide ID for segment, define view of data base
             segment, define pseudo segment, define mapping segment,
             define output segment

   FIELD     Define field storage layout, display format, audit
             status; define keywords or text as part of output
             segment

   FIELD     Second field in first segment group

     .                    .

     .                    .

     .                    .

   FIELD     Last field in first segment group

 SEGMENT     Second segment for data base definitions, set parentage
             to establish data base path and hierarchy in path

   FIELD     First field in second segment group

     .                    .

     .                    .

     .                    .

   FIELD     Last field in second segment group

 SEGMENT     Last segment for application system for this run

   FIELD     First field in last segment definition

     .                    .

     .                    .

     .                    .

   FIELD     Last field in last segment definition

GENERATE     Request Sign-on Screen and Primary Option Menu

GENERATE     Request rules and screens for particular segments

GENERATE     Request rules and screens for transaction

GENERATE     Request Secondary Option Menu Rule

GENERATE     Request transaction driver link-edit
```

Figure 2-1. Rules Generator Control Statement Hierarchy

Note: INCLUDE statements may appear at any point within the input stream.

Input to the Rules Generator is one or more segment definitions followed by requests to generate static rules and screens. The static rules, screens, or load modules created from a Rules Generator run depend upon the GENERATE statement specifications. These specifications are determined by the current transaction requirements, plus previously generated rules, screens, and modules. For example, if a Segment Layout Rule and Segment Handler Rule have already been generated for a segment they need not be regenerated for each transaction.

Figure 2-2 and Figure 2-3 indicate the frequency of rule or screen generation and the operands required to create them.

RULE	OPTIONS OPERAND	FREQUENCY
Primary Option Menu	POM	One for each major application system identification
Secondary Option Menu	SOM	One for each major application system identification
Input Transaction	INTR TPIT BAIT TEXT	One for each transaction ID and one for each Project/Group that will process the transaction
Segment Layout	SEGL	One for each segment specified in an Input Transaction Rule
Mapping Segment Layout	SEGL	One for each group of fields requiring mapping between Special Processing routine or user-written audit exit routine and segment storage areas
Output Format Layout	SEGL	One for each secondary transaction or output message to be sent
Segment Handler	SEGH	One for each segment in the data base

Figure 2-2. Rules Generation Frequency and OPTIONS Operands

SCREEN	OPTIONS OPERAND	FREQUENCY
Sign-on	SIGN	One for each major application system identification
Primary Key Selection	KEYD	One for each Conversational Input Transaction Rule for regular or text segments
Segment Display	SEGD	One for each Conversational or Nonconversational response Input Transaction Rule
Text Utility	TEXT	One for each text segment
Secondary Transaction Output/Routed Message	OMFS	One for each Output Format Rule requesting IMS MFS or CICS BMS formatted output message. (Destination may be a program, a screen or a printer.)

Figure 2-3. Screen Generation Frequency and OPTIONS Operands

For details of Rules Generator keyword usage with rule and screen definitions, refer to Appendix A, "Summary of Rules Generator Operand Usage."

APPLICATION DEFINITION AND RULES GENERATOR EXECUTION EXAMPLE

All the rules for an application system are placed in a rule library of the user's choice using a JCL procedure variable that is the prefix part of a rule load library naming convention. The RULES parameter is used for this purpose. The rule library naming conventions can be changed in the distributed procedure to reflect installation conventions conventions. To illustrate how easy it is to use the Rules Generator, the following paragraphs describe an application that requires processing of one data base segment in the conversational mode.

An application system inventory DL/I data base contains a 50-byte root segment. The DBD sequence or key field is the 17-byte part number and starts in position 1 of the segment. The only other field to be seen in the online display application is the part description, which starts at byte 27 of the segment and is 20 characters long. The Rules Generator statements required to provide rules and screens for an application that can add new part numbers to this application data base are shown in Figure 2-4.

This set of control statements will cause one Segment Layout, Segment Handler, and Input Transaction Rule to be generated for the 'PA' segment. In addition, a Key Selection screen and a Segment Display screen will be generated. Since this is the first transaction of a new application system, a Sign-on screen and a Primary Option Menu Rule also are requested. Finally, a Secondary Option Menu Rule is generated to include the IMSADF II transaction 'P1'.

```
//USER      JOB      STATEMENT
//STEP1     EXEC     ????G,RULES=LIB1
//G1.SYSIN  DD       *
          SYSTEM     SYSID=PART,DBID=PA,PGROUP=CC,SOMTX=OR,
                     SHEADING='PART INVENTORY SYSTEM'
          GENERATE   OPTIONS=(SIGN,POM)
          SEGMENT    ID=PA,LENGTH=50,NAME=PARTROOT,KEYNAME=PARTNUMB
            FIELD    ID=KEY,LENGTH=17,KEY=YES,SNAME='PART NUMBER'
            FIELD    ID=DESC,LENGTH=20,POS=27,
                     SNAME='PART DESCRIPTION'
          GENERATE   OPTIONS=(SEGL,SEGH),SEGMENT=PA
          GENERATE   OPTIONS=(INTR,KEYD,SEGD),TRXID=P1,DBPATH=PA,
                     TRXNAME='PART DESCRIPTION'
          GENERATE   OPTION=SOM
/*
```

Figure 2-4. Example of Segment Definition and Rules Generation Job Submittal

CODING AND SYNTAX CONVENTIONS

Each input card or card image can contain control statement information in columns 1 through 71. Columns 73 to 80 can be used for user sequence numbering. In cases where the statement operand ends in column 71, a blank must be in column 72. Otherwise, column 72 can be used as part of the user comment string on a statement. The Rules Generator builds assembler source code for ASMR=YES, thus the assembler character set should be used when defining keyword data as Rules Generator input source. The ALPHANUMERIC set plus special characters @, # and $ may be used. The Rules Generator uses commas, periods and underscores in parsing keywords and values input.

The following conventions must be observed when coding the control statements for the Rules Generator.

1. The control statement type keyword can start in any card column and must be followed by one or more blanks. These keywords can be abbreviated to two characters.

2. The operands must be separated by commas with no intervening blanks and can be coded in any order.

   Each operand and its operand value or values must be contained on a single card. Continuation to a new card cannot break between an operand and its value or between values.

3. If more than one value follows the operand keywords, these values must be separated by commas and the entire value specification must be enclosed in parentheses.

4. The operand keywords can be abbreviated to a minimum set of characters that uniquely defines each keyword belonging to a statement type.

5. Each statement can be continued on as many cards as necessary in the following manner. The card to be continued must terminate with a comma followed by a blank. The following card can start in any column. A blank immediately following an operand value terminates the processing of that statement.

6. When character data contains embedded blanks, it must be enclosed in single quotation marks (' '). Text data cannot contain quotes within the text.

7. Comments can be used on any card as long as they are preceded by the one or more blanks that follow the operand text.

8. An asterisk in column one of any card will cause that card to be treated as a comment.

9. When an operand has multiple values or option selections, all must be entered on the same card. If more values are necessary than can be entered on one card, the keyword or keywords must be repeated on a series of cards. For example, two cards containing

   OPT=SEGH,

   OPT=SEGL

   would be the same as one card containing OPT=(SEGH,SEGL).

10. Underscore characters may be present as operand values. If an operand value only contains underscores on entry to the Rules Generator, that is the same as if the operand keyword was not entered. If data is entered for an operand, leading and trailing underscore characters (and commas as required) are removed before processing.

    For example:

```
SEGMENT     ID=XX,NAME=DBDN1___,BYTES=_60_,PCBNO=___

  ...

GENERATE    OPT=CVALL,DBPATH=(A1,A2,__),
            TRX=AA,TRXN='_____NAME__),
```

    is equivalent to the following:

```
SEGMENT     ID=XX,NAME=DBDN1,BYTES=60

  ...

GENERATE    OPT=CVALL,DBPATH=(A1,A2),
            TRX=AA,TRXN='NAME'
```

RULES GENERATOR CONTROL STATEMENTS

This section describes the six Rules Generator Control statements (SYSTEM, SEGMENT, FIELD, GENERATE, RULE and INCLUDE) in detail. For each control statement, a summary table of all the operands for that statement is presented first, followed by a more detailed description of the function and use of each operand. The description of each operand is listed in alphabetical order, whereas the operands in the summary tables are grouped by function.

For the purpose of describing the specification of these operands, the operands can be viewed as belonging to two groups: those that have default values and those that do not have default values. Operands without default values can be further divided into required and optional operands. The operands with default values are always optional.

The following syntax convention is used to describe the operands:

- If an operand has a default value, the default value is underlined in the operand value list. An * indicates that there exists a default whose value depends on the value of some other operand.
- A vertical operand value list enclosed by braces { } means that one value must be selected from the list. A horizontal operand value list enclosed by braces means one or more values may be specified from the list.
- An optional operand without a default value has brackets [ ] enclosing the operand and its operand value list. Thus, an operand without brackets and without a default value is a required operand.
- The minimum set of characters needed to identify an operand uniquely is shown below the keyword in the summary table.
- Upper case letters should be entered as written. Character variables are represented by 'x'; numeric variables are represented by 'n'. If a variable is referred to by another operand, it will be referred to by a variable name, noted by lower case letters (e.g. segid). Literal values are noted by quotations.
- An ellipsis (...) indicates that the preceding item or group of items may be repeated.

SYSTEM STATEMENT

The SYSTEM statement defines the major application system identification characters. It also defines other common information and default values that can be used to build rules and screens. The default options and common information for building the rules and screen source are obtained from the SYSTEM statement unless either the SEGMENT or the GENERATE statement contains operands that override those in the SYSTEM statement. SYSTEM statement defaults which can be overridden include those that specify the user data base identification and PCB number, and automatic screen control and heading.

A SYSTEM statement must precede any segment definitions in the input stream. Multiple SYSTEM statements can be supplied in one input stream. Each time a SYSTEM statement is encountered, the Rules Generator overrides the previous system control information. Multiple system statements may be used in a single rules generation run by placing the transactions back to back and specifying RESET=YES on the SYSTEM statement. The RESET=YES parameter clears the Rules Generator data areas and does not override the previous SYSTEM statement parameters. If the major system ID is changed or RESET=YES is specified, the Rules Generator data area or segment transaction is initialized to zero. The operand summary for the SYSTEM statement is shown in Figure 2-5.

OPERANDS	VALUES	DESCRIPTION
Transaction and Data Base Identification		
ADFID= AD	{MFC1}* {xxxx}	Application Development Facility ID
AGROUP= AG	{YYYY} {xxxx}	audit group ID
DBID= DB	{DB} {xx}	data base ID
KANAME= KAN	{ALT}* {STD}	key audit access under the root key
PCBNO= PC	{1} {n}	PCB number in PSB
PGROUP= PG	xx	project group ID
SOMTX= SOMT	xx	IMS/VS transaction cluster code
STRAILER= ST	{1} {x}	screen name trailer
SYSID= SY	xxxx	major application system ID
USRLANG= US	x (A-Z, 0-9) *	language to be associated with SYSID
Rules and Screen Generation		
ENVIR= EN	{CICS}* {IMS}	BMS or MFS source generation
[POMENU= POME	{ALL ({A,B,C,D,F,H,I})}]	Primary Option Menu generation
SIGNON= SI	{NO RACF YES}	Sign-on screen generation
SOIMAGE= SOI	{NO YES}	screen image used for Sign-on
Screen Format and Content		
SFORMAT= SF	{DASH LEFT RIGHT}	screen format for literals
[SHEADING= SHE	'1 to 54 chars']	screen heading
MAXKEY= MAX	{50} {n}	screen key length

* see parameter description for default value

Figure 2-5 (Part 1 of 2). SYSTEM Statement Summary

OPERANDS	VALUES	DESCRIPTION
Rules Generator Output and Interface		
ASMLIST= ASML	{LIST NOGEN NOLIST}	request for type of listing
ASMNAME= ASMN	{IEV90 IFOX00} *	-equest for specific assembler
DECKS= DEC	{NO YES}	request for rule source deck
LINEC= LI	{58 n}	number of lines in listing
[LOPTPARM= LO	'1 to 36 chars']	Linkage Editor options
LRULE= LR	{NO ALT YES}	request for retrieving static audit rules
POMDECK= POMD	{NO YES}	request for primary option menu source deck
RESET= RE	{NO YES} *	request for clearing segment tables
SSI= SS	{0000 mmyy xxxx}	four-character Linkage Editor information

* see parameter description for default value

Figure 2-5 (Part 2 of 2). SYSTEM Statement Summary

OPERANDS (in alphabetical order):

ADFID= {**MFC1**} {**XXXX**}

The IMSADF II identification operand is used to specify with which installation of IMSADF II the currently generated rules must function. The default value is set at installation time. The operand value must be four unique characters. (See the IMS Application Development Facility II Version 2 Release 2 Installation Guide.)

AGROUP= {**YYYY**} {**XXXX**}

The AGROUP operand specifies the last four characters of the Audit Group name contained in the Input Transaction Rule. This name is used to identify specific Audit/logic routines connected with a transaction(s). The format of the Audit Group name is ssssxxxx where,

ssss = the major application system identification specified in SYSID

xxxx = any four characters to identify the Audit Group in the Audit Data Base

The default value is YYYY. Each AGROUP operand specified on a GENERATE statement overrides the value specified for AGROUP on the SYSTEM statement.

ASMLIST= {**LIST**} {**NOGEN**} {**NOLIST**}

The ASMLIST operand specifies whether or not an output listing is required from a rule assembly. This operand is only applicable when the Assembler is invoked either as a request using the ASMREQ operand for the batch transaction driver or special processing link-edits. The value NOLIST indicates that no listing is required. The value LIST indicates that a listing is required. The value NOGEN indicates that a listing is required, but that macro generated Assembler source is suppressed. The default value is LIST. This operand specified on a GENERATE statement overrides the value specified on the SYSTEM statement.

ASMNAME= {**IEV90**} **(Assembler H)** {**IFOX00**} **(Assembler F)**

The ASMNAME operand specifies the Assembler to be used if ASMREQ=YES is specified or if the OPTION=BDLE or OPTION=PREL is used. The default for ASMNAME is the installed value in ADFOPTNS.

DBID= {**DB**} {**XX**}

The DBID operand specifies the two-character data base identification assigned to the physical or logical data base that is described in the subsequent SEGMENT and FIELD statements. DBID is used to interrelate the Input Transaction Rule and the Segment Layout Rule. The DBID default value is DB. When this operand is specified on a SEGMENT statement, its value overrides the value specified for DBID on the SYSTEM statement.

DECKS= {**NO**} {**YES**}

The DECKS operand may be used to request an Assembler Language form of application rules. When DECK=YES is specified, the generated Assembler Language rule source will be output to a data set with

DDNAME=GENDECK. Each rule is automatically preceded by a RULE statement for subsequent invocation of the Assembler as required.

The DECKS=NO operand value which is the default specifies that object decks are to be created and link-edited in the appropriate rules load library. The Preload rule and the batch transaction driver and special processing link-edits make use of the Assembler during rules generation. The segment and transaction related rules are created directly in object by the Rules Generator unless the ASMREQ=YES operand value is also specified, unless the ASMREQ=YES operand value is also specified, in which case Assembler source statements are built and the Assembler is invoked dynamically to produce object modules. Object decks created directly or by the Assembler are placed on a temporary file for link-editing at the end of the rules generation process. If rules are to be placed in more than one rules library the LINKREQ and LINKLIB operands must be used.

Note that there are no Assembler listings for segment and transaction related rules when the ASMREQ=YES operand has not been specified. See the ASMREQ and ASMLIST operand descriptions for information on Assembler listings. A listing of the Linkage Editor execution is provided. Each DECKS operand specified on a GENERATE statement overrides the value specified for DECKS on the SYSTEM statement.

ENVIR= {**CICS**|**IMS**}

The ENVIR operand specifies the screen source to be generated will be either CICS/OS/VS BMS or IMS/VS MFS. The default is the DEFADF ENVIR specification set at installation time.

KANAME= {**ALT**|**STD**}

The KANAME operand is used to specify whether Key Audits in the transactions are to be accessed under the Root Key 'SSSSYYYYSSXXFFFF' when KANAME=ALT is used. If the current naming convention of 'KEYAUDITSSXXFFFF' is desired then KANAME=STD may be specified. The default value for KANAME is the installed value in ADFOPTNS. Refer to "Audit Logic Rules" in Chapter 3, "Audit Language"

LINEC= {**58**|**n**}

The LINEC operand specifies the maximum number of lines per page for the Rules Generator output listing data set that is defined by DDNAME=GENLIST. The default value is 58. The Rules Generator also accepts the keyword EJECT as the first non-blank characters of a card or card image as a command to eject to a new page and set the current line count to zero.

[LOPTPARM='1 to 36 chars']

The LOPTPARM operand is used to override the Linkage Editor options on a rules generation or module link-edit. Default values are LIST,REUS,XREF,MAP,SIZE=(256K,64K). A maximum of 36 characters may be specified.

LRULE= {**NO**|**ALT**|**YES**}

The LRULE operand is used to specify how the audit rule data is to be retrieved. A YES value indicates that the audit descriptors are to be retrieved from a Static Audit load module, which is loaded by Audit Group name, at the start of execution. The value will be generated as part of each Input Transaction Rule. Specification of the LRULE operand of the GENERATE statement will override the system value for the current GENERATE statement. The default is NO, which means that the audit descriptors are to come from the dynamic Audit Data Base. A value of ALT specifies that the static

rule should be retrieved and executed if possible. If the static rule cannot be found or specific audits are not present in the rule, the Audit Data Base is accessed for the dynamic rules. LRULE=YES will yield maximum performance in the auditor.

MAXKEY= {**50** | **n**}

The MAXKEY operand may be used to set the length of the screen area for entering key data directly in the KEY field on the segment display screen. The default is 50 characters. For non-screen image input a MAXKEY value of 50 or less will result in the SYSMSG field being placed on line 5. A MAXKEY value greater than 50 will result in a SYSMSG field being placed on line 2 so that KEY: may use a contiguous field of up to 100 characters. For screen image input (SPOS=SIMAGE operand of GENERATE statement) the current MAXKEY length will be used to determine if any screen fields overlap. If &KEY is not specified in a screen image, a default of 50 will be used and the KEY: field will be placed on last screen line. The maximum key length is 100.

PCBNO= {**1** | **n**}

The PCBNO operand is a number from 1 to 120 that indicates the relative application data base PCB within the PSB that is used to access the hierarchical path or paths described in the subsequent SEGMENT statements.

The PSBs used by transaction drivers have IMSADF II dynamic rules data base PCBs included in front of the user's application data base PCBs. The Rules Generator will automatically account for IMSADF II data bases. The default value is 1. Each PCBNO operand specified on a SEGMENT statement overrides the value specified for PCBNO on the SYSTEM statement.

PGROUP=xx

The PGROUP operand specifies the two-character Project/Group to be used in building the Input Transaction Rule name for conversational transactions. If this operand is not specified on the SYSTEM statement, each GENERATE statement that requests an Input Transaction Rule must provide the specification for Project/Group. When this operand is specified on a GENERATE statement, its value overrides the value specified for PGROUP on the SYSTEM statement.

POMDECK= {**NO** | **YES**}

The POMDECK operand specifies whether or not a rule source deck is to be created for the Primary Option Menu Rule. The value YES indicates that the generated Assembler language Primary Option Menu Rule source will be written to a data set with DDNAME=GENDECK. NO is the default value and indicates that an object deck will be created and link-edited in the rules load library.

[**POMENU=** {**ALL** | **({A,B,C,D,F,H,I})**}]

The POMENU operand indicates that a Primary Option Menu Rule is to be generated and specifies which options are to be displayed on the Primary Option Menu screen. A value of ALL indicates that all options A through I are to be displayed. Combinations of options can be specified. The options represent the following areas of processing:

A Project Message Sending
B Project Message Display
C Session Termination
D Transaction Selection
F Project/Group Switch
H User Message Sending
I User Message Display

Refer to the IMS Application Development Facility II Version 2 Release 2 User Reference for an example of the uses of the various options. If the POMENU operand is entered on the SYSTEM statement, the Primary Option Menu Rule is generated following the SYSTEM statement. The POMENU operand is optional. Note that the GENERATE statement may also specify this operand. However, it will be ignored on the GENERATE statement if it was already specified on a previous SYSTEM or GENERATE statement.

RESET= {**NO** | **YES**}

The RESET operand is used to clear segment tables for a new system. If this operand is not specified and the SYSTEM statement is used to change data base or screen information within a major application system (SYSID), segment tables will not be reset. If the operand is not specified and the SYSTEM statement specifies a new SYSID, previous segment definitions will be cleared automatically. The RESET operand can be specified to override the default values.

SFORMAT= {**DASH** | **LEFT** | **RIGHT**}

The SFORMAT operand defines the format of the literal fields to be generated for the Primary Key Selection and Segment Display screens. When screen image statements are provided for defining a transaction, the SFORMAT operand is not used for Segment Display screen generation (see SPOS=SIMAGE operand value of the GENERATE statement for screen image literal specifications). The Rules Generator builds a screen literal and variable field for each FIELD statement requiring display on a Primary Key or Segment Display screen. For Primary Key Selection, the display fields are the key fields specified with KEY=YES. For Segment Display, the display fields are those fields specified with DISPLAY=YES. Each literal is built from the value of SNAME as it is specified in the appropriate field definition. Screen centering is calculated based on the length of both the longest SNAME literal and the longest displayed field.

SFORMAT defines how the literal fields will appear on the screen. The value DASH specifies that dashes are to be added to make each literal length equal to the longest SNAME length plus one. The value LEFT indicates that the literal is left justified and followed by blanks. The value RIGHT indicates that the literal is right justified and preceded by blanks. The default value is DASH. This operand specified on a GENERATE statement overrides the value specified on the SYSTEM statement. An example of the DASH operand value is as follows:

```
LITERAL 1------- data
LIT 2----------- data
LONGEST LITERAL- data
```

An example of the LEFT operand value is as follows:

```
LITERAL 1       data
LIT 2           data
LONGEST LITERAL data
```

An example of the RIGHT operand value is as follows:

```
      LITERAL 1 data
          LIT 2 data
LONGEST LITERAL data
```

Left and right justification applies to the literal area of an aligned screen when the automatic screen layout option is used for the Segment Display screen or for the Primary Key Selection screen. The length of the literal area of an aligned screen is the length of the longest literal to be displayed. When explicit field

positioning is specified for the Segment Display Screen (SPOS=ROWCOL), the SFORMAT options apply to an area length between the two columns specified. If only one column is specified, the Rules Generator will use right justification for that field without regard to the SFORMAT option. If two rows are specified, one for the literal and one for the data area, the Rules Generator will use left justification.

[SHEADING='1 to 54 chars']

The SHEADING operand contains the text to be displayed on the first line of the Sign-on screen, Primary Key Selection screen, Segment Display screen, and Text Utility Screen. SHEADING text is enclosed in single quotation marks (' ') and can be from 1 to 54 characters in length. The text becomes a literal in the screen source statements generated for the requested screens. The default value is one blank. This operand specified on a GENERATE statement overrides the value specified on the SYSTEM statement.

SIGNON= { **NO** | **RACF** | **YES** }

The SIGNON operand specifies whether or not a Sign-on screen is to be generated. The value YES indicates that screen source is to be generated and written to the data set pointed to by DDNAME=SCREENS. The four-character major application system identification specified in the SYSID operand is the Sign-on screen MOD name. The default value is NO. Note that the GENERATE statement can also be used to request a Sign-on screen (but not on both statements). If Sign-on screens are desired for terminals other than those specified as installation defaults, the GENERATE statement must be used. In those cases, the DEVNAME and DEVTYPE operands of the GENERATE statement must also be specified. The value of RACF will cause the Sign-on screen to be generated without the USERID field and during sign-on processing the RACF USERID is used as the Sign-on USERID. The following statement will build the Sign-on screen with a RACF interface:

```
System SYSID=SAMP,PGROUP=TL,PCBNO=1,SIGNON=RACF
```

SOIMAGE= { **NO** | **YES** }

SOIMAGE=YES indicates that a Sign-on screen is to be built using the screen image which immediately follows this statement. With SOIMAGE=YES on either the SYSTEM or GENERATE statements, you may not also request a Sign-on screen in both places.

If SOIMAGE=NO and a Sign-on screen is requested, a default screen will be generated. The default is SOIMAGE=NO.

The fields allowed on the Sign-on screen image are:

1. Literals - Literal data is entered directly in the screen image statements at the required screen locations.

2. System fields - Fields of the form &adfname may be used where adfname is a system field as described below. (The CURSOR operand on the GENERATE statement may refer to these fields for cursor positioning.)

   USERID six-character userid from the Sign-on profile. This field is optional for SIGNON=RACF.

   PROJECT one-character project identifier from the Sign-on profile.

   GROUP one-character group identifier from the Sign-on profile.

   LOCKWORD eight-character password.

OPTION one-character option as would normally be chosen from the primary option menu.

TRAN transaction mode and secondary option value. Format will be mss, where m is the current mode number chosen for the transaction and ss is the secondary option value, which can be either a target segment or a Special Processing ID.

KEY concatenated key of the hierarchical path being displayed. Maximum length for the KEYID field on the Sign-on screen is 50 bytes. Key length cannot be altered by MAXKEY.

SYSMSG 70-character system message field. This field will be displayed in high intensity and protected.

SYSID four-character System ID, may eliminate the need for a different Sign-on screen for each system.

LTNAME eight-character terminal identifier.

TIME eight-character format HH:MM:SS (current time).

DATE1 six-character format YY.DDD (current date).

DATE2 eight-character format MM/DD/YY (current date).

DATE3 eight-character format DD/MM/YY (current date).

DATE4 eight-character format YY/MM/DD (current date).

The first eight system fields are required. If they are not specified, an attempt will be made to default them to the last three lines as follows:

```
Last Line     : SYSMSG.
Last Line - 1 : OPTION,TRAN,KEYID.
Last Line - 2 : USERID,PROJECT,GROUP,LOCKWORD.
```

If defaulting any of these fields causes an overlap with any other field on the screen, a terminating error message will be generated.

The last seven system fields are optional. The last six system fields are not applicable if ENVIR=CICS is specified on either the SYSTEM or GENERATE statements.

3. Control fields - Fields of the form &control specify screen image control parameters, where control is described below. In each case the &must be entered in column 1 and must be immediately followed by the control character.

   &* - Specifies that this is a comment statement. Comments follow the asterisk.

   &=nn - Specifies the number of blank lines to insert on the screen before the next line is created.

Position 1 of line 1 is reserved and not available to the user. Positions 1 through 4 of the last line are reserved for the size field and not available to the user.

The Sign-on screen will have the following default display attributes for these system fields:

field	default display mode
USERID	5 (display/modifiable)
PROJECT	5 (display/modifiable)
GROUP	5 (display/modifiable)

LOCKWORD 7 (non-display/modifiable)

Mode 7 may be specified for USERID, PROJECT, and GROUP system fields which causes these fields to be built as Non-Display/Modifiable. Mode 5 may also be specified for LOCKWORD which causes this field to be built as Display/Modifiable.

EXAMPLE -

```
        GENERATE OPTIONS=SIGN,SOIMAGE=YES,
                 SHEADING='BANKING SYSTEM'
               &DATE2                                &TIME
                                                     &SYSID
               BANKING SYSTEM

      ENTER YOUR 6 CHARACTER USERID:-------------------- &7USERID
      ENTER YOUR 1 CHARACTER PROJECT NUMBER:------------ &PROJECT
      ENTER YOUR 1 CHARACTER GROUP NUMBER:-------------- &7GROUP
      ENTER YOUR 8 CHARACTER PASSWORD--IF APPLICABLE:--- &5LOCKWORD

          OPTIONS                          TRANSACTION MODES

A = PROJECT MESSAGE SENDING                   3 - REMOVE
B = PROJECT MESSAGE DISPLAY                   4 - ADD
C = SESSION TERMINATED                        5 - UPDATE
D = TRANSACTION SELECTION                     6 - RETRIEVE
F = PROJECT GROUP SWITCH                FOR OPTION - IDENTIFIER IS
H = USER MESSAGE SENDING                      D    - TRANSACTION ID
I = USER MESSAGE DISPLAY                      F    - PROJECT GROUP
                                         A,B,C,H,I - (NOT USED)

      TRX IS THE CONCATENATION OF TRANSACTION MODE AND IDENTIFIER
      OPTION:       TRX:       KEY:
&ENDS
```

In the above example, the system fields USERID and GROUP will be defined as Non-Display/Modifiable. The LOCKWORD will be defined as Display/Modifiable.

SOMTX=xx

The SOMTX operand specifies the two-character IMS/VS transaction cluster code that is to define the IMS/VS transaction, the PSB, and the application load module names. This operand is required for the Secondary Option Menu Rule. This allows multiple IMSADF II transaction IDs to be associated with one IMS/VS transaction. The SOMTX operand specified in a GENERATE statement will override this value if multiple transaction IDs, with different cluster codes, are created in one Rules Generator run.

SSI= { 0000 | mmyy | xxxx }

The SSI operand specifies information which will be used in a Linkage Editor SETSSI statement to set the system status index of the directory entry when a rule or transaction driver load module is link-edited. If mmyy is specified, the month and year are placed in the first four characters of the SETSSI statement in the link-edit. Four EBCDIC characters are required if this operand is used. The other four EBCDIC characters are reserved for IMSADF II usage. Refer to the Operating System Linkage Editor manual for additional information on the SETSSI command.

STRAILER= { 1 | x }

The STRAILER operand value character is used to produce unique generated names for screen formats. It is used in conjunction with the ADFID operand so that a unique set of screens may be used with each installed version of IMSADF II. The default value may be set at installation time. One alphanumeric character may be entered.

SYSID=xxxx

The SYSID operand specifies the four-character major application system identification. The generated rules will contain the SYSID in the first four characters of the rule name. If screen source is to be generated for the Sign-on screen, the SYSID value will be used as the Message Output Description (MOD) name. The SYSID operand is required.

USRLANG=x

The USRLANG keyword on the System Statement defines the language to be associated with that SYSID. If USRLANG= is not specified, the Rules Generator will use the default language specified at install time. If USRLANG= is specified, the Rules Generator will load the specified MFC1G40 if it is not already in storage and will use that module to build rules and screens for the end user. The following example will cause the German language module to be loaded for SYSID=INFO:

```
SYSTEM ADFID=MFC1,SYSID=INFO,SOMTX=01,USRLANG=G
```

The following USRLANG values are allocated to the corresponding language:

USRLANG Value	Language Module
E	English
F	French
G	German
J	Japanese
K	Korean
P	Portuguese
S	Spanish
W	Swedish

Other USRLANG values of A-Z and 0-9 are available for other languages.

SEGMENT/TABLE STATEMENT

The SEGMENT statement is used to define a data base segment. Each application segment definition includes a SEGMENT statement followed by one or more FIELD statements. A SEGMENT statement or a GENERATE statement following a FIELD statement terminates the segment definition that precedes it. The OVERRIDE operand may be used to search for a previously defined segment and modify the segment statement operands. In this case, field definition statements do not follow the segment statement. However, segment qualified field override statements (ID=ffff.ss) may follow any override SEGMENT statement. A segment definition can describe a particular view of an application data base segment, define a set of pseudo segment fields, or define a group of fields for mapping, or Output Format Rule generation. For DB2 support TABLE and SEGMENT keywords are interchangeable. The TABLE statement describes a DB2 TABLE/VIEW.

The TYPE operand is used to define the type of segment; i.e., PS for pseudo segment, MAP for map, OUT for an output segment, and DB for data base segment definition. The default segment type operand is DB. IMSADF II can have different views of the same data base segment by defining a different set of fields following a SEGMENT statement with a different ID, but specifying the same segment name (NAME=data base segment name).

Pseudo segments are defined to provide segment work area(s) for the transaction. These work areas may be required to hold data values which do not relate to a data base segment field. Pseudo segments are used to hold screen or transaction input, audit calculations, or Special Processing routine calculations. Specification of a pseudo segment will cause space to be reserved in the SPA segment area and the Segment Layout Rule to be loaded at the start of the transaction. Fields in a pseudo segment may be displayed and updated in the same manner as a data base segment.

Mapping segments describe other segment fields which will be moved between the SPA segment area and a Special Processing routine or a user-written audit exit. Fields may also be mapped within the SPA using the audit language MAP statement. When defining a mapping segment, the field ID (ID=) and segment ID (SEGID=) specified point to segment field(s) previously defined in the Rules Generator run.

Output segments define the layout of a secondary transaction which may be sent by the transaction driver. This definition, known as an Output Format Rule, describes fields, literals, and keyword values which will comprise the contents of the secondary transaction.

Each segment-type described is generated into a Segment Layout Rule format through the GENERATE operand OPTION=SEGL. The required SEGMENT statement operands vary according to the type of segment. The required operands for each type of segment is shown below. If bytes are skipped, field definitions overlap, or bit data is specified, the POSITION operand is required. Data base segment definitions must include:

```
ID      =   unique two-character segment identification
LENGTH  =   segment length
PARENT  =   0    (specified for data base root segments)
PARENT  =   segment ID or NAME of parent (required for non-root
                segments)
PCBNO   =   data base PCB number if the number differs from that
                on the SYSTEM statement
NAME    =   DBD name of segment
KEYNAME =   DBD sequence field name of segment
```

- At least one field in the data base segment definition must be defined as a key field. The total length of key fields in one segment can not exceed 70 bytes.

Pseudo segment definitions must include:

```
ID      =   unique two-character segment identification
TYPE    =   PS
```

- If not specified, segment length is determined from the sum of the field lengths

Mapping segment definitions must include:

```
ID      =   unique two-character segment identification
TYPE    =   MAP
```

- Each field statement in a mapping segment must have a SEGID operand
- If not specified, segment length is determined from the sum of the field lengths

Output segment definitions must include:

```
ID      =   unique two-character segment identification
TYPE    =   OUT
```

- Field statements in an output segment may be used to enter a text string or reference an IMSADF II keyword in addition to referencing a data base field or pseudo segment field. In the latter case SEGID must be specified.

```
FIELD  ID=yyyy,SEGID=xx,...
        or
FIELD  KWNAME=zzzzzzzz,...
        or
FIELD  TEXT='text string or trancode'
```

- If not specified, segment length is determined from the sum of the field lengths.

SEGMENT statements must appear after a SYSTEM statement. The PARENT operands indicate the level of a segment in the data base to the Rules Generator. More than one path can be described to the Rules Generator through the use of the PARENT operand. More than one data base can be described to the Rules Generator by using the PARENT=0 operand value for the root segment of each data base.

The operand summary for the SEGMENT statement is shown in Figure 2-6.

OPERANDS	VALUES	DESCRIPTION
Segment Identification		
[DBID= DB	xx]	data base ID
ID= ID	segid	two-character IMSADF II segment ID
LENGTH= LEN or BYTES= BY	n n	segment length
NAME= NA	segname	eight-character DBD name. This operand is required for data base segments only
NEWID= NE	segid	new two-character ID to replace existing segment ID on segment override
OVERRIDE= OV	NO ID NAME YES	segment statement is to override previous statement
PARENT= PA	0 segid segname	parent ID or name; 0 for root only data base, DB2 table, or VSAM file
[PCBNO= PC	n]	PCB number in PSB
TRAILER= TR	01 xx	two-character trailer for data base name
TYPE= TY	DBS COMM DBEK ESDS KSDS MAP OUT PS RRDS RREK TBL	segment type
Segment Processing		
[DCFIELD= DC	fieldid]	field to validate prior to update

Figure 2-6 (Part 1 of 2). SEGMENT Statement Summary

OPERANDS	VALUES	DESCRIPTION
Screen Formats and Content		
DISPLAY= DI	{NO }* {YES}	default display for fields in segment
[MODE= MO	n]	default mode for fields in segment
[SKLEFT= SKL	'1 to 36 chars']	header for Secondary Key Selection screen
[SKRIGHT= SKR	'1 to 36 chars']	header for Secondary Key Selection screen
[SKSEGS= SKS	n]	number of segments for Secondary Key Selection screen
Key Related Specification		
KASCEND= KA	{YES} {NO }	key values ascend or are unordered
KEYNAME= KEYN	xxxxxxxx	DBD sequence field name
DB2 Related Specifications		
SQLDIST= SQLD	{NO } {YES}	DB2 distinct flag
SQLIND= SQLI	{NO } {YES}	indicator variable flag
SQLISRT= SQLIS	{YES} {NO }	column insert flag
SQLNAME= SQLN	'1 to 50 chars'	DB2 TABLE/VIEW name
SQLUPD= SQLU	{YES} {NO }	column update flag

* see parameter description for default value.

Figure 2-6 (Part 2 of 2). SEGMENT Statement Summary

OPERANDS (in alphabetical order):

BYTES=n
or
LENGTH=n

The BYTES or LENGTH operand specifies the length of the physical or logical data base segment. The maximum value that can be specified for this operand is 6000 bytes. The value specified is placed in the Segment Layout Rule. The BYTES or LENGTH operand is required only for data base segment definitions.

[DBID=xx]

DBID specifies the two-character data base identification that is assigned to the physical or logical data base described by the SEGMENT statement. This operand overrides the DBID operand specified in the SYSTEM statement.

[DCFIELD=fieldid]

The DCFIELD operand operates in conjunction with the DATACOMP operand, on the GENERATE statement, to provide field validation prior to the update of a segment. If multiple users can update the same segment concurrently in Conversational processing, the segment should be checked to insure that the data base version and the version displayed on the screen in the SPA area are the same. DATACOMP will cause this compare to be made. If only a single field is critical, then that field xxxx can be specified on the DCFIELD operand, and then, only that field is compared prior to the segment update. The field ID must be a field defined within the segment.

DISPLAY= {**NO** | **YES**}

The DISPLAY operands on the FIELD and SEGMENT statements are not used for generation requests that provide screen image statements for defining the transaction. See SPOS=SIMAGE operand value of the GENERATE statement for screen image field display specifications.

The DISPLAY operand indicates whether the segment is to be displayed on the segment display screen when that segment is part of an Input Transaction Rule. If this operand is set, each field's display attribute will be set to the DISPLAY value of the SEGMENT statement. However, this value will be overridden by each DISPLAY operand specified on a FIELD statement.

The Display operand used in generation of Batch or Nonconversational Input Transaction rules specifies the segment fields which make up the input transaction data. See the FLDPOS operand on the FIELD statement for further information about this specification.

Defaults are set depending on whether a segment defined in an Input Transaction Rule is a target segment or a non-target segment. The default for a data base target segment is YES. For a segment in the path leading to the target segment, the default is NO. The default for a pseudo segment is YES.

ID=segid

ID specifies the two-character identification associated with data base, pseudo, mapping, and output segments. It is used by IMSADF II to interrelate rules. ID is a required operand.

KASCEND= {**YES** | **NO**}

A value of YES indicates that this data base segment's keys are in physical ascending order (1,2,3, etc). This operand applies when this segment is named in the DBPATH operand (explicitly or implied due to a displayable field) and the 'N' option is entered on the segment display screen, or the 'F' option is used on the secondary

key selection screen. The Conversational transaction driver will use this operand to determine which DL/I calls will be used to position into the data base for the next screen. (KASCEND=Y ----> GU > current key value, KASCEND=N ----> GU = current key value, GN).

Note that if a delete has been performed against this segment and a subsequent 'N' option is entered, the data base position technique must be GU > deleted key. This will cause unpredictable positioning if keys are not in ascending order (KASCEND=NO).

KEYNAME=xxxxxxxx

KEYNAME specifies the DBD sequence field name for this segment as it is defined to IMS/VS. It must be specified if the DBD sequence field is made up of more than one Rules Generator FIELD statement, or the NAME operand is not specified on the FIELD statement defining the sequence field. The name is one to eight characters in length and is placed in the Segment Handler Rule.

If the segment does not have a unique sequence field defined in the DBD see "Processing Segments with Non-Unique Keys and No Keys" in Appendix D, "Additional Application Support Techniques" for a description of the supported functions and processing methods capable of handling the various situations.

LENGTH=n
or
BYTES=n

The LENGTH or BYTES operand specifies the length of the physical or logical data base segment. The maximum value that can be specified for this operand is 6000 bytes. The value specified is placed in the Segment Layout Rule. The LENGTH or BYTES operand is required only for data base segment definitions.

[MODE=n]

The MODE operand specifies the mode each field in this segment will have as a default. The MODE operand only applies to fields that result in a DISPLAY=YES attribute. The modes that can be specified are:

4 (Modifiable, modifiable on transaction mode 6)
5 (Modifiable)
6 (Non-Modifiable)
7 (Non-Display, Modifiable)

Note that the segment mode specification will override the target/non-target defaults. Specification of the MODE operand for any field within a segment will override the segment MODE operand default.

The default for all non-key fields in a Standard Processing Segment Display screen is mode 5 (modifiable). The default for all key fields is mode 6 (non-modifiable). The default for all data base segment fields in a Special Processing Segment Display screen is mode 6 (non-modifiable). The default for pseudo segment fields is mode 5 (modifiable).

The MODE operands on the SEGMENT and FIELD statements are not used for generation requests that provide screen image statements for defining the transaction. See SPOS=SIMAGE operand of the GENERATE statement for screen image mode specifications.

NAME=segname

The NAME operand is the DBD segment name as it is known to IMS/VS. This name, which is one to eight characters in length, is placed in the Segment Handler Rule and is used for DL/I calls. This allows data base segments to be accessed by DL/I with one name and to be identified to IMSADF II with the character identification in the ID operand. This operand is required for data base segments if user

naming standards are followed instead of IMSADF II naming standards.

NEWID=segid

NEWID specifies the two-character segment ID which is to override the ID parameter on a segment override. If search is on ID then that ID may be changed by specifying NEWID. See example under the OVERRIDE operand description for use of this keyword.

OVERRIDE= { **NO** / **ID** / **NAME** / **YES** }

The OVERRIDE operand is used to specify that this SEGMENT statement is to override an existing segment definition. The previously entered SEGMENT statement will be found by either an ID or NAME search. The OVERRIDE=NO, OVERRIDE=YES, and OVERRIDE=ID operand values each specify that a search for a previously defined segment with the ID specified on this SEGMENT statement is to be performed. The default value, OVERRIDE=NO, will cause the previous segment definition with the same ID to be replaced. The OVERRIDE=YES and OVERRIDE=ID operand values which are equivalent, specify that SEGMENT operands are to be overridden but the definition is not to be replaced. The OVERRIDE=NAME operand value specifies that a search for a previously defined segment with the NAME specified on this SEGMENT statement is to be performed. Operands that appear on the override SEGMENT statement will replace corresponding operands on the previous entry. Operands that cannot be overridden are LENGTH or BYTES, DISPLAY, and MODE. If search is by NAME, an ID operand will replace the previous ID. If search is by ID, a NAME operand will replace the previous NAME. The Segment ID may be changed by doing a search by ID and specifying the new segment ID with the keyword NEWID. Overridden values remain in effect unless changed by another override SEGMENT statement.

Field definitions do not follow override SEGMENT statements. However, field override statements with segment qualified field IDs (ffff.ss) may be used immediately following this SEGMENT statement.

This is illustrated in the following example:

```
*BASIC SEGMENT DEFINITION

 SEGMENT     ID=AA,LEN=50,PAR=0,NAME=AAAA,
             KEYNAME=BBBB,PCBNO=1
   FIELD     ID=A1,LEN=20,KEY=YES,SNAME=CARDKEY
   FIELD     ID=A2,LEN=7,AUDIT=YES,
             SNAME='OWNER NUMBER'

*OVERRIDE SEGMENT

 SEGMENT     OVER=ID,ID=AA,PCBNO=3,
             SKLEFT='KEY NUMBER',
             SKRIGHT='OWNER NUMBER'
   FIELD     ID=A1.AA,KAUDIT=YES,COL=7
   FIELD     ID=A2.AA,RELATED=YES,COL=47
 GENERATE    OPT=CVALL,DBP=AA,TRXID=T1,
             TRXNAME=NEWDATA,SOMTX=CC

*BUILD ALIAS SEGMENT

 SEGMENT     ID=AA,LEN=50,PAR=0,NAME=AAAA,
             KEYNAME=BBBB,PCBNO=1
   FIELD     ID=A1,LEN=20,KEY=YES,SNAME=CARDKEY
   FIELD     ID=A2,LEN=7,AUDIT=YES,
             SNAME='OWNER NUMBER'
 SEGMENT     OVER=ID,ID=AA,NEWID=AB
```

PARENT= { **0** / **segid** / **segname** }

The PARENT operand is used to define the parentage of data base segments. The segid is the two-character ID for the parent segment of this segment. A PARENT=0 specifies that this is a root segment. Segname specifies the eight-character DBD name of the parent segment. This is defined through the NAME= operand. The Rules Generator uses each PARENT operand value to determine the data base path for Key Selection, Segment Handler, and Input Transaction Rule generation. This operand is required for all non-root data base segments. If the PARENT=segname used, the segment that is referred to must precede the segment that refers to it and the segment name must be unique. If PARENT=segid is used, the segments may be specified in any order.

[PCBNO=n]

PCBNO is a number from 1 to 120 that indicates the relative data base PCB within the PSB. This operand overrides the PCBNO operand specified in the SYSTEM statement.

[SKLEFT='1 to 36 character literal']
[SKRIGHT='1 to 36 character literal']

The SKLEFT and SKRIGHT operands specify the Secondary Key Selection screen header text line for a segment. If these operands are not specified, the program will generate a heading based on the maximum size of each field literal for those fields that are to appear on the Secondary Key Selection screen. If either or both operands are specified, the COLUMN operand specified in the FIELD statement can be used to locate the key and related field data below the 72-character heading (combined SKLEFT and SKRIGHT). For example, specifying a COL=1 entry for a field would place the data for that field below the first character entered in the SKLEFT literal. A COL=37 entry for a field would place the values for that field below the first character entered in the SKRIGHT literal. If the number of characters entered is less than 36 in either literal, left justification within the appropriate heading area will occur. SKLEFT and SKRIGHT specified twice will produce two lines of heading. The first character of the heading starts on column 9 of the Secondary Key Selection screen.

[SKSEGS=n]

The SKSEGS operand of the SEGMENT statement is used to specify the number of segment occurrences to be displayed on the Secondary Key Selection screen for this segment. If more occurrences than this value exist, the specified number of segments are displayed and a 'F' option is displayed to indicate that there are more occurrences. The default value for root segments is 0; meaning that only Primary Key Selection is available (the key value must be known to access the data base). The default for non-root data base segments is a function of the device size. If a second heading line is defined, this maximum is decreased by one before rules generation.

If the SKSEGS operand value is specified with a value greater than 0 for a root segment, then Secondary Key Selection is available. The SKSEGS operand value may be entered as 0 for non-root data base segments to only allow primary key selection.

The maximum number of lines available is 37 (which is for a 43 line device). If SKSEGS is greater than the number of available lines on any device, the available lines are used instead of SKSEGS. For segments with non-ascending key values for the field marked KEY=YES, the KASCEND parameter should be specified. Secondary Key Selection of root segments can then be invoked on the Primary Key Selection screen by entering a greater than (>) character in the first position of the key field. If a portion of the key value is entered followed by a greater than character, then the first segment that has a key field greater than the key portion followed by binary zeros will be the starting segment for the Secondary Key

Selection screen. For non-ascending keys, use of the generic key (>) will produce unpredictable results.

The first segment selection option is also available for all non-root segments. A portion of the key followed by a greater than character can be entered in each key field in the Primary Key Selection screen. If multiple key fields make up a DBD key for a segment level, only that portion of the key that precedes the first greater than character entered will make up the generic key.

For DB2, SKSEGS=0 indicates that no browsing is available for this table/view. SKESEGS=n specifies the number of row occurrences to be displayed. It indicates that this DB2 table/view will be processed by the Key Selection Browse function.

SQLDIST= {**NO** | **YES**} **(DB2 only)**

SQLDIST indicates that the SQL DISTINCT keyword will be used in the SELECT SQL statements (both standard SQLCALL and user specified SQLUSER) generated in the Table Handler Rule.

SQLIND= {**NO** | **YES**} **(DB2 only)**

SQLIND sets a flag in the Table Layout Rule Column header and sets a flag in each Table Layout Rule Column definition to indicate that an indicator variable is associated with each column in this Table/View. Indicator variables are available to receive the SQL indication that a column is NULL or truncated or to set a column NULL in the UPDATE or INSERT statements.

Indicator variables, if present, will be used by screen handling for NULL displays and can be tested or set (NULL) in audits. Each indicator variable is a binary halfword. The table I/O area length reflects the length of the columns and the indicator variables.

Note that IMSADF II will generate indicator variables for all columns but will use the indicator variable (SQL statements and audits) only for those columns, specifying SQLNULL=YES.

Format of the Table I/O area as defined by the Table Layout Rule follows:

Table Definition Information
Data Elements (Columns) COL1 . . . . . . COL2 . . . . . . COLn . . . . . .
Indicator Variables IND1 . . . . . . IND2 . . . . . . INDn . . . . . .

SQLISRT= {**YES** | **NO**} **(DB2 only)**

SQLISRT=YES indicates that all Columns in the Table are eligible for insert. SQLISRT=NO indicates that no Columns are eligible for insert.

Specifying SQLISRT=YES

- causes SQLISRT=YES to be specified for each Column entry.
- includes the Column in the VALUES clause of the SQL INSERT statement in the Table Handler Rule.

Specification of SQLISRT on the FIELD statement will override the SQLISRT operand on the SEGMENT statement.

SQLNAME='1 to 50 characters' (DB2 only)

The SQLNAME operand specifies the Table/View name(s) as specified in the DB2 catalog. The NAME will be in a qualified or unqualified format and will contain as many names as will fit in 50 characters. Multiple names are used to define a Join. If qualified, a period separates the authorization ID and the Table name. Name formats are the following:

```
Unqualified name--Table name (long identifier - 1 to 18 char)
Qualified name----Authorization ID.Table name
                   where Auth ID is a short identifier (1 to 8 char)
                   and Table name is a long identifier (1 to 18 char)
Join--------------'table1, table2, table3'
```

The SQLNAME operand is required to build the Table Handler Rule to define SQL access to DB2.

SQLUPD= {YES / NO} (DB2 only)

SQLUPD=YES indicates that all Columns in the Table are eligible for update. SQLUPD=NO indicates that no Columns are eligible for update.

Specifying SQLUPD=YES

- Causes SQLUPD=YES to be specified for each Column entry, (except KEY=YES Columns).
- includes the Column in the SET clause of the SQL UPDATE statement in the Table Handler Rule.

Examples of columns that cannot be updated are those derived from built-in functions or arithmetic expressions. Other columns not eligible for update are application dependent.

Specification of SQLUPD on the FIELD statement will override the SQLUPD operand on the SEGMENT statement.

TRAILER= {01 / xx}

TRAILER is a two-character trailer for completing the IMSADF II Segment name used to interrelate Input Transaction Rule, and the Segment Layout Rule. The default value is 01.

TYPE= {DBS / COMM / DBEK / ESDS / KSDS / MAP / OUT / PS / RRDS / RREK / TBL}

The TYPE operand is used to specify whether the segment is a data base segment, a pseudo segment, a mapping segment, or an output segment.

The DBS value designates a data base segment definition. When this value is specified, a segment layout generation will use the characters 'SR' in the fifth and sixth positions of the rule name. The segment length operand is required if this value is specified. DBS is the default value.

The COMM value indicates that the storage area for a pseudo segment will reside in the SPA communications area instead of the SPA

segment I/O area in a conversational transaction. The user then may access the SPA communications area with audit operations. For example:

```
*
* PSEUDO SEGMENT TO MAP COMMUNICATIONS AREA
*
  SEGMENT   ID=CS,LEN=36,TYPE=COMM                <---
    FIELD   ID=OPT1,TYPE=NUM,LEN=6
    FIELD   ID=PGU2,TYPE=ALPHA,LEN=6
    FIELD   ID=FLD3,LEN=16
```

This segment definition will be added to an IMSADF II transaction as would a standard pseudo segment.

```
*
* PSEUDO SEGMENT TO MAP COMMUNICATIONS AREA
*
  GENERATE  TRXID=CA,DBPATH=CA,OPTION=CVALL,
            ISRT=CR,TRXNAME='CHECK-ACCNT ',
            COMMLEN=50,TSEG=(PS,CS)               <---
```

In nonconversational and batch transactions, COMM segments are processed as standard pseudo segments and are placed in the segment I/O area since no communications area exists in these modes.

The DBEK value designates that the segment is a data base segment with an external key. This operand supports FAST PATH Processing. This operand will be used to set a flag in the Segment Handler Rule to be used by Fast Path Processing.

The MAP value designates a mapping segment definition. When this value is specified, a segment layout generation uses the characters 'UR' in the fifth and sixth positions of the rule name. The segment length operand is not required if this value is specified. FIELD statements following this segment definition must each contain a SEGID operand. If the TYPE operand of a mapping field is different from the TYPE operand of the referenced data base or pseudo segment field, data conversion will take place as described in Appendix B, "Data Conversions and Mappings."

The OUT value designates an output segment definition. When this value is specified, a segment layout generation uses the characters 'OR' in the third and fourth positions of the rule name. The segment length operand is not required if this value is specified. Field statements referring to data base or pseudo segment fields must specify both ID and SEGID operands. If the TYPE operand of an output segment field is different from the TYPE operand of the referenced data base or pseudo segment field, data conversion will be performed according to Appendix B, "Data Conversions and Mappings." Neither ID nor SEGID operands are required for text or keyword field statements (TEXT or KWNAME operands).

The PS value designates a pseudo segment definition. Pseudo segments are defined to be used as work areas. These areas are used read and display data. The Segment Layout Rule name will contain the 'SR' characters.

The TBL value is added to the TYPE operand to indicate a DB2 Table/View is being defined. If the TABLE statement is used (rather than SEGMENT), the default value for TYPE is TBL.

The values ESDS, KSDS, RRDS, and RREK are VSAM file types which are processed as though they were data base segments. The Segment Handler Rule will contain a flag to indicate which segment type was specified.

FIELD/COLUMN STATEMENT

FIELD statements describe the attributes of data within a segment. For DB2 support, Column and Field Keywords are interchangeable. The categories into which these data attributes fall are:

- Attributes that interrelate rules and fields
- Attributes that describe data base, pseudo segment, mapping or output layouts
- Attributes that describe the key selection process
- Attributes used to generate display screens
- Attributes that describe audit criteria

The first category allows IMSADF II to use field IDs to interrelate rules. The second category includes the attributes that specify LENGTH, POSITION, KEY, NAME, TYPE, DECIMAL or BITOFF, and COMPARE and VALUE for storage and retrieval. Attributes that describe the key selection process include KAUDIT, KDISPLAY, KEYBLD, COFIELD, RELATED, and COLUMN. A status parameter confirming the accepted combinations of these attributes is also provided (see KSELECT). The fourth category includes the attributes that determine the fields and literals to be displayed on a screen. For Segment Display screens, these attributes are SNAME, DISPLAY, MODE, SLENGTH, SDECIMAL, SROW, and SCOL. For Primary key selection screens these are SNAME (either specified field or COFIELD) and KDISPLAY. For Secondary Key Selection screens, these are SNAME, RELATED, and COLUMN. The fifth category includes the attributes that specify whether a field is to be audited and whether automatic message sending is required. These attributes are AUDIT, CAUDIT, PAUDIT, MSG, AFA, FAUDIT, and REQUIRED. An audit status parameter combining these seven attributes into one keyword and value is also provided (see ASTATUS).

Fields may be redefined with different attributes by specifying different field IDs. For example, a field 15 bytes in length may be defined and then redefined as 3 fields 5 bytes in length.

```
FIELD ID=xxxx,POS=10,LEN=15
FIELD ID=yyyy,POS=10,LEN=5
FIELD ID=zzzz,POS=15,LEN=5
FIELD ID=wwww,POS=20,LEN=5
```

Fields with the same ID or name following a segment statement will be merged. The later specifications override earlier ones, except for the POSITION or START operand for which the first specification holds.

```
FIELD ID=xxxx,POS=10,LEN=10,REQ=YES
FIELD ID=xxxx,DISP=Y,REQ=NO
            equals
FIELD ID=xxxx,POS=10,LEN=10,REQ=NO,DISP=Y
```

Fields may be initially defined in a segment with no KEY=YES attribute and later be redefined as KEY=YES by specifying ID=ffff.ss where ffff is field ID and ss is segment ID. When redefining these fields as KEY=YES, the field must be the DBD sequence field as defined in the NAME or KEYNAME operands. Fields defined as KEY=YES cannot be redefined as KEY=NO. If the DL/I DB sequence field is divided into several IMSADF II key fields, then the IMSADF II key fields must be coded in DB order.

```
SEGMENT   ID=AA,LENGTH=130,PARENT=0,TYPE=DBS
FIELD     ID=ABCD,LEN=2,POS=1
FIELD     ID=EFGH,LEN=8,POS=3
SEGMENT   ID=BB,LEN=130,PARENT=AA,TYPE=DBS
FIELD     ID=IJKL,LEN=2,POS=1
FIELD     ID=MNOP,LEN=8,POS=3

FIELD     ID=ABCD.AA,KEY=YES
FIELD     ID=IJKL.BB,KEY=YES
GENERATE
```

If field definition overrides are desired for fields in previously defined segments, it may be specified by ID=ffff.ss, where ffff is one field ID and ss is one segment ID. These field override statements are effective only if no segment definition is currently specified; i.e., when a GENERATE statement has been specified following the last segment definition. For example:

```
INCLUDE SEGMENT=(AA,BB,CC)
GENERATE
FIELD ID=xxxx.AA,PAUDIT=YES
GENERATE OPTIONS=SEGL,SEGMENT=AA
```

will result in field xxxx of segment AA having preaudit status turned on.

The operand summary for the FIELD statement is shown in Figure 2-7.

OPERANDS	VALUES	DESCRIPTION
Field Identification and Description		
BITOFF= BI	{0 {n	bit offset for BIT type data
DECIMAL= DE	{0 {n	number of digits to the right of decimal
[FLDOL= FLDO	[L][O][R][U]]	field outlining for KANJI 5550 terminal
ID= ID	{fieldid {fieldid.segid	IMSADF II field identification
KEY= KE	{NO {YES	key field
LENGTH= LEN or BYTES= BY	n n	field length
[NAME= [NA	{fieldname {(xxxxxxxx,SEQ,...)	DBD field name
[OFFSET= OFF	n]	specifies displacement from start of field entered for redefine
[POSITION= PO or [START= ST	n] n]	field position in segment
[REDEFINE= RED	fieldid]	specify previous defined field for start of redefine sub-field
SIGN= SI	{YES {NO	decimal or packed decimal field plus or minus an indicator in stored value
TYPE= TY=	ALPHANUM C ALPHA A BIN I BIT B DATE DA or D DBCS DB DEC DE or Z FLOAT F HEX H or X MIXED M NUM N PD P UDEC UD or UZ UPD UP VARCHAR V	data type

Figure 2-7 (Part 1 of 4). FIELD Statement Summary

OPERANDS	VALUES	DESCRIPTION
Screen Display		
DISPLAY= DI	{NO {YES } *	request for display
[MODE= MO	n]	field modifiable or not
[DMODE= DM	{ N { Y { n { Yn { Nn }]	combines DISPLAY and MODE operands
[SDECIMAL= SD	n]	screen decimal position
[SLENGTH= SL	n]	field screen length
[SNAME= SN	'1 to 30 chars']	screen name for field
[SROW= SR	{n {(n,n) }]	SNAME row and field data row
[SCOL= SC	{n {(n,n) }]	SNAME column and field data column
Key Selection		
COFIELD= COF	{fieldid.segid {fieldname }	name of pseudo field to hold displayable form of field
[COLUMN= COL	n]	field column position on Secondary Key Selection screen
KDISPLAY= KD	{YES {NO }	request for a display of a key field during key selection phase
KEYBLD= KEYB	{fieldid.segid {fieldname }	key value
KSELECT= KSEL	{ N { K { KA { KB { KC { KN { KP { KS { R { RC { RS }	specifies key selection phase operation
RELATED= REL	{NO {YES }	request for display on Secondary Key Selection screen

* see parameter description for default value

Figure 2-7 (Part 2 of 4). FIELD Statement Summary

OPERANDS	VALUES	DESCRIPTION
Audit Requests		
AFA= AF	NO YES	request for AFA audit
AUDIT= AU	NO YES	request for field audit
CAUDIT= CA	NO YES	request for common audit
FAUDIT= FA	NO YES	request for force audit
KAUDIT= KA	NO PRIM SECO YES	request for key selection phase audit
MSG= MSG	NO YES	request for message sending
PAUDIT= PA	NO YES	request for preaudit
REQUIRED= REQ	NO YES *	request for required audit
[ASTATUS= AS	[A][C][F][M][P][R][Y]]	request for any or all audits
Batch and Nonconversational Input		
[FLDPOS= FL	n]	field input position
ILENGTH= IL	n	field input length
ITYPE= ITY	ALPHANUM C ALPHA A BIN I BIT B DATE DA or D DBCS DB DEC DE or Z FLOAT F HEX H or X MIXED M NUM N PD P UDEC UD or UZ UPD UP VARCHAR V	field type

* see parameter description for default value

Figure 2-7 (Part 3 of 4). FIELD Statement Summary

OPERANDS	VALUES	DESCRIPTION
Special Processing		
[COMPARE= COM	{EQ GE GT LE LT NE}]	compare operand
[VALUE= VA	'1 to 30 chars']	compare value
Output Format or Mapping Rule		
[KWNAME= KW	xxxxxxxx]	keyword field specification
RDONLY= RD	{NO YES}	read only status for mapping segment
SEGID= SEGI	segid	data base or pseudo segment for field
[TEXT= TE	'1 to 30 chars']	text string for output segment
DB2 Related Specifications		
SQLISRT= SQLI	{YES NO}	column insert flag
SQLORD= SQLO	{ASC DESC}	ORDER of Columns with KEY=YES
[SQLNAME= SQLNA	'1 to 50 chars']	DB2 Column VIEW NAME
SQLNULL= SQLN	{NO YES}	column null indicator
SQLUPD= SQLU	{YES NO}	column update flag

Figure 2-7 (Part 4 of 4). FIELD Statement Summary

OPERANDS (in alphabetical order):

AFA= {NO / YES}

A YES value in the AFA operand builds an FFRUL macro in the Input Transaction Rule for this field and sets AFA=YES. The value YES specifies that the Auditor must check the Automatic Field Assignment leg of the Audit Data Base for audit/logic instructions. The default value is NO.

[ASTATUS=[A][C][F][M][P][R][Y]]

The ASTATUS operand may be used to specify all audit operands (AUDIT, CAUDIT, MSG, REQUIRED, FAUDIT, PAUDIT, or AFA) in one operand value. This keyword operand will set the various parameters to YES or NO. Including the key character sets the operand to YES. Not specifying the key character sets the operand to the default value.

Character Code	Operand Setting
A	AFA=YES
C	CAUDIT=YES
F	FAUDIT=YES
M	MSG=YES
P	PAUDIT=YES
R	REQUIRED=YES
Y	AUDIT=YES

Refer to the Auditor section in Chapter 3, "Audit Language" for additional information on audit parameters.

AUDIT= {NO / YES}

The AUDIT operand sets the AUD operand in the Segment Layout Rule to specify auditing requirements. A field will be audited through the Field Audit leg of the Audit Data Base if AUDIT=YES and the field has been changed or is specified as KEY=YES.

BITOFF= {0 / n}

The BITOFF operand specifies the offset of a bit in a byte when TYPE=BIT. Valid bit offsets are 0 to 7. The default value is 0.

BYTES=n
or
LENGTH=n

The BYTES or LENGTH operand specifies the length of a field as it is to be stored in the segment. The maximum value that can be specified for this operand is 255 bytes. The field length is stored in the Segment Layout Rule. The operand is also used to generate the Key Selection screen for data base segment fields and the Input Transaction Rule and Segment Display screen for both data base and pseudo segment fields. If the SLENGTH operand is not specified, the display lengths will be expanded by the Rules Generator on the basis of storage length whenever the field data requires conversion for display.

A DBD sequence field can be divided into more than one FIELD statement for display flexibility. The total length of these key fields must equal the DBD sequence field length. The maximum length DBD sequence field that can be processed by IMSADF II is 70 bytes. The BYTES or LENGTH operand is required for all fields in DBS, PS, and MAP segment definitions. The BYTES or LENGTH operand is not required for TEXT or KWNAME fields in output segment definitions. If one of these operands is not specified for TEXT fields, the text string length including blanks will be used. If it is not specified for KWNAME fields, the length of the keyword area in storage will be used.

The range of valid lengths by field type are:

TYPE	LENGTH
Alphabetic	1-255
Alphanumeric	1-255
Binary	2 or 4
Bit	1
Date	6
DBCS	2-256 (must be even number of characters)
Decimal	1-15
Float	8
Hexadecimal	1-255
Mixed	4
Numeric	1-255 (1-15 if used for Audit arithmetic)
Packed Decimal	1-8
Varchar	1-253

CAUDIT= { **NO** | **YES** }

The CAUDIT operand specifies whether or not the field name is common throughout the installation and has common Audit requirements. The default is NO.

COFIELD= { **field.segid** | **fieldname** }

COFIELD is specified on key fields and Secondary Key Selection related fields. It names a pseudo segment field that will hold the displayable form of that field on the Primary and Secondary Key Selection screens. COFIELD is specified only when a data base key field or data base related field must be displayed to the terminal operator in a form different from the normal data base to screen conversions. For example, a key field that is in an encoded form and is displayed to the terminal in a decoded table look-up form; complemented keys in the data base and uncomplemented to the screen. COFIELD for a key field requires both a primary and secondary KAUDIT to perform the special conversion. COFIELD for a related field requires a secondary KAUDIT to perform the special conversion. When a key field names a COFIELD to hold the displayable form of the field and that pseudo segment field is changed on the Segment Display, this is treated as any other DBPATH key change and a request for new occurrences.

The use of the generic key (>) is not processed for a COFIELD. If attempted, an error message is output. If COFIELD is specified, the KEYBLD operand does not apply.

Note: If COFIELD= is specified on a field statement, the Rules Generator will look for a pseudo segment, previously defined, with the field ID specified. This means that the field ID referenced in the COFIELD= must have been defined in the Rules Generator Source statements prior to the field definition that has COFIELD= keyword. If the pseudo segment has not been processed already, an ADF031 error message will result.

Note: The use of COFIELD requires forward and backward pointers between pseudo segment fields and data base fields. The design allows a user to reference different fields of the same pseudo segment as a cofield in more than one Input Transaction Rule in a rules generation run, but not the same pseudo segment field. An ADF031 message results if the same pseudo segment field is referenced more than once as a cofield in a rules generation run.

[COLUMN=n]

The COLUMN operand specifies the column number within the 72-byte secondary header text line on the Secondary Key Selection screen at which this key or related field begins. The positions that can be specified are 1 to 72. If COLUMN is not specified, the Rules Generator calculates the next available column plus 2 bytes as the first field location. This operand works in conjunction with the

SKLEFT and SKRIGHT secondary screen heading operands and is not used if neither SKLEFT or SKRIGHT is specified.

```
[COMPARE= { EQ }
          { GE }
          { GT } ]
          { LE }
          { LT }
          { NE }
```

The COMPARE operand adds the DBD non-sequence field and the specified Boolean Operator to the SSA in the Segment Handler Rule. The compare value is specified with the VALUE operand. When a NAME operand has been specified, the field name is obtained from it. Otherwise, the field name is generated in the ssxxzzzz format.

The COMPARE operand is specified when a Special Processing Routine has a Segment Handler requirement to retrieve a segment qualified with a secondary compare. The COMPARE operand is not specified for Standard Processing.

```
DECIMAL= { 0 }
         { n } (1-13)
```

When TYPE=DEC or PD, this operand specifies the number of digits to the right of the decimal point. The DECIMAL operand may also be specified for an alphanumeric mapping or output segment field that is to be converted from a TYPE=DEC or PD field. The default value is 0.

```
DISPLAY= { NO  }
         { YES }
```

The DISPLAY operand indicates whether or not a field is to be displayed on the Segment Display or included as input transaction data in Batch and Nonconversational processing. The field will be identified by an ITRFLD statement in the Input Transaction Rule and will be placed in the screen source for the Segment Display screen.

Defaults are set depending on whether a segment defined in an Input Transaction Rule is a target segment or a non-target segment. This is described under DISPLAY for the SEGMENT statement. This operand overrides the DISPLAY operand specified in the SEGMENT statement.

```
        { N  }      (equivalent to DISPLAY=NO)
        { Y  }      (equivalent to DISPLAY=YES)
[DMODE= { n  } ]    (Mode=n)
        { Yn }      (equivalent to DISPLAY=Y, MODE=n)
        { Nn }      (equivalent to DISPLAY=N, MODE=n)
```

The DMODE operand of the field statement is used to combine the DISPLAY and MODE operands. This operand provides a single tabular positioning when using the tabular form of field definition. A maximum of two characters may be entered.

```
FAUDIT= { NO  }
        { YES }
```

A YES value in the FAUDIT operand builds an FFRUL macro in the Input Transaction Rule for this field and sets FAUD=YES. A value of YES specifies that the field should be marked as changed by the Auditor. Auditing will be unconditionally forced on a field specified with FAUDIT=YES and AUDIT=YES. The default value is NO.

[FLDOL=[L][O][R][U]]

The FLDOL operand may be used to select field outlining for the 5550 Terminal (KANJI).

Character Code	Operand Description
L	Left line for field

O Overline for field length

R Right line for field

U Underline for field length

[FLDPOS=n]

The FLDPOS operand specifies the relative position of a field in the input area, which contains an input transaction message. This operand is used for Nonconversational (without SPOS=SIMAGE) and Batch processing. For online transaction processing, the first field position is 13 to account for the eight-character IMS/VS transaction name, one blank, the transaction mode, and the transaction ID. Each subsequent specification must account for the length of the previous field. The Nonconversational Transaction Driver accounts for the IMS/VS LL, Z1, and Z2 fields. For batch processing, the first FLDPOS is 9 to account for the eight-character transaction name. For variable length records, the first FLDPOS is 11. The DISPLAY=YES operand must be specified or defaulted for the field before FLDPOS operand will be used. If not present, the FLDPOS operand is ignored. No default is assumed. If included for Nonconversational and SPOS=SIMAGE is specified, the FLDPOS operand is ignored.

ID= { **fieldid** | **fieldid.segid** }

The ID operand value fieldid is one to four characters in length. It provides a unique identifier for a field within a segment. The inter-rule naming convention for fields is ssxxzzzz where

ss = major application system identification

xx = ID of segment

fieldid = one to four meaningful characters identifying the field

The ID operand is required for all fields in DB, PS, and MAP segment definitions. The ID operand value fieldid.segid is used for field override statements. When the field statement is not immediately following another field statement or a SEGMENT statement the segid portion of the ID operand is used to locate the previous segment for which a field override is required. The ID operand is required for fields in OUT segment definitions that refer to data base or pseudo segment fields. The ID operand in MAP and OUT segment definitions points to a previously defined field definition. In addition, the SEGID= keyword is required for MAP and OUT segment definitions.

ILENGTH=n

The ILENGTH operand is used to specify the field length as supplied in the input area of a nonconversational or batch input transaction message. If SIMAGE is used in nonconversational processing then ILENGTH is ignored and a SLENGTH value is calculated. If SIMAGE is used in BATCH mode and ILENGTH is specified, the ILENGTH value will be used for field length. If SIMAGE is not used and ILENGTH and ITYPE are not specified then ILENGTH will be calculated the same as SLENGTH ILENGTH will be calculated the same as SLENGTH (this implies that the input stream is card image, and that data conversion may be required). If SIMAGE and ILENGTH are not used but ITYPE is specified then the storage length will be used for ILENGTH (this implies a transaction to transaction message where no data conversion is required).

```
        ALPHANUM        C    (alphanumeric characters)
        ALPHA           A    (alpha characters)
        BIN             I    (binary number)
        BIT             B    (bit data)
        DATE      DA or D    (input form mm/dd/yy)
        DBCS            DB   (double byte character set)
        DEC       DE or Z    (zoned decimal)
ITYPE=  FLOAT           F    (floating point)
        HEX        H or X    (hexadecimal representation)
        MIXED           M    (mixed EBCDIC and DBCS)
        NUM             N    (numeric characters)
        PD              P    (packed decimal)
        UDEC      UD or UZ   (unsigned zoned decimal)
        UPD             UP   (unsigned packed decimal)
        VARCHAR         V    (variable character)
```

The ITYPE operand is used to specify the field input type when a nonconversational or batch Input Transaction Rule is requested. The default value is ALPHANUM. Certain restrictions apply in defining the input data. Input data fields must be long enough to accommodate the displayable form of the data. Data will be converted and adjusted where reasonable. The conversion chart in Appendix B, "Data Conversions and Mappings" defines the conversion that will be attempted.

```
KAUDIT= { NO   }
        { PRIM }
        { SECO }
        { YES  }
```

KAUDIT applies only to key fields or related fields in DBPATH segments.

This operand specifies that a key audit is to be performed on this key or related field. Refer to Chapter 3, "Audit Language" for a description of the Key Audit phase.

A value of PRIM indicates the key is read from a screen (Conversational or Nonconversational) or from batch input. The key is converted to its data base form and the Auditor is invoked to perform a Key Audit for this key field. If the key field is not displayed (KDISPLAY=NO), then the field will be initialized to its data base value prior to invoking the Auditor. Upon successful return from the Auditor, the DL/I call will be made.

A value of SECO indicates that during Secondary Key Selection, the Auditor is invoked after each retrieval of a segment for display on the Secondary Key Selection screen. KAUDIT will apply to key or related fields. On return from the Auditor, the segment will be eligible for display or it will be skipped and not shown on the Secondary Key Selection screen. When a selection is made from the Secondary Key Selection screen, Key Audit is again invoked to get the displayable form of the key.

A value of YES indicates both PRIM and SECO.

```
KDISPLAY= { YES }
          { NO  }
```

The KDISPLAY operand specifies whether this key field is displayed on the Primary Key Selection screen or Secondary Key Selection screen and in the key header (KEY:keys). It applies to DBPATH segment keys. If KDISPLAY=NO, the key should be built by the developer using KAUDIT or KEYBLD processing. The default value is YES.

```
KEY= { NO  }
     { YES }
```

The KEY operand specifies whether or not the field is the DBD sequence field. More than one FIELD statement that follows a SEGMENT statement can specify KEY=YES, and at least one must specify KEY=YES for data base segment definitions. However, the total length for all fields specified in FIELD statements with

KEY=YES must match the DBD sequence field length defined to IMS/VS. The maximum length DBD sequence field that can be processed by IMSADF II is 70 bytes. The default value is NO. (See KSELECT). If the DL/I DB sequence field is divided into several IMSADF II key fields, then the IMSADF II key fields must be coded in DB order.

For DB2, the KEY operand will specify those Columns to be used in the WHERE expression of the SQL statement for the Table Handler Rule. The relational operator between the DB2 column name and the IMSADF II host variable is '=' for the SELECT, DELETE, and UPDATE statements. The standard Secondary Key Selection browse SELECT uses '>=' or '<=' or LIKE (character strings only) for its relational operator.

IMSADF II standard processing functions assume that the key values uniquely identify a row. It is recommended that KEY=YES columns are columns named in the Table CREATE INDEX for better DB2 performance in in processing the WHERE statement.

If more than one column indicates KEY=YES and a composite INDEX exists, the order of the columns presented to the Rules Generator should match the order specified in the CREATE INDEX.

Every table must have at least one column specifying KEY=YES to uniquely identify a row. This requirement exists for all data base segments (DL/I and DB2) that are defined to IMSADF II. If the table is accessed through one of the IMSADF II standard SQL calls, the table's KEY=YES columns will be included in the WHERE clause. Even though a TABLE may be accessed only through SQLUSER WHERE statements or native SQL calls that do not reference the KEY=YES columns in the WHERE clause, this requirement is still in effect.

TYPE=FLOAT and TYPE=BIT will not be processed as a key.

KEYBLD= { **fieldname** | **fieldid.segid** }

The KEYBLD operand specifies a field in a previously defined segment definition which contains the key value for retrieving this segment. This operand is used in multiple path definitions when a path is loaded and from that path the key of a segment in another path can be retrieved. KEYBLD implies that the key value will not be entered through Primary or Secondary Key Selection. This operand is valid only for fields specified as KEY=YES or KSELECT=KB. Field name is a one- to eight-character DBD name of the source field described by a NAME= operand. The source field can also be specified by a one- to four-character field ID and a two-character segment ID. The segment containing the source field must be loaded prior to loading the KEYBLD segment and in a different hierarchical path. (See DBPATH operand for loading information.) The COFIELD operand is invalid for a KEYBLD field.

KSELECT=

N	**non-key/not related**
K	**key field/display**
KA	**key field/display/primary audit/secondary audit**
KB	**key build field/no display/no audits**
KC	**key cofield/primary audit/secondary audit**
KN	**key field/no display/primary audit**
KP	**key field/display/primary audit**
KS	**key field/display/secondary audit**
R	**related field**
RC	**related cofield/secondary audit**
RS	**related field/secondary audit**

The KSELECT operand specifies the key selection phase operation to be performed on this field. The available options represent the acceptable combinations of key building and related operations during the key selection process. For example, when the KEYBLD parameter is specified, no display or auditing during the key selection phase is performed. Specifying COFIELD requires that key selection phase auditing be performed to process the related input data. The values KA, KN, KS, RC, and RS are only used for Conversational transactions. The KSELECT operand is used in place

of the KEY, KDISPLAY, KAUDIT, and RELATED operands when the tabular form of field specification is used.

[KWNAME=xxxxxxxx]

The KWNAME operand specifies that a field is a keyword field from the IMSADF II Driver SPA or Common Segment Handler COMOPT area. This operand is valid only for TYPE=OUT segments. The following is a list of valid SPA and COMOPT keyword names and the default data type and length that will be generated in the output format rule:

SPA FIELDS	TYPE	LENGTH	COMOPT FIELDS	TYPE	LENGTH
			COSOURCE	BIT	1
SPAMANNO	ALPHANUM	6	COMOD	BIT	1
SPAUSER	ALPHANUM	11	COSEQ	BIT	1
SPAPROJ	ALPHANUM	1	COSEGR	BIT	1
SPAGROUP	ALPHANUM	1	COMSTAT	ALPHANUM	2
SPATRX	ALPHANUM	3	COTRANS	ALPHANUM	3
SPATRXCD	ALPHANUM	1	COOPER	ALPHANUM	1
SPATRXSG	ALPHANUM	2	COSEGM	ALPHANUM	2
SPAKEYID	ALPHANUM	255	COKEY	ALPHANUM	100
SPAERMSG	ALPHANUM	8			
SPASHOTR	ALPHANUM	8			
SPALEGTH	BIN	2			
SPATRANS	ALPHANUM	8			
SPASYSID	ALPHANUM	4			
SPADATE	ALPHANUM	5			
SPASIGON	ALPHANUM	6			
SPALTERM	ALPHANUM	8			
SPACGTRX	ALPHANUM	3			
SPATXEBC	ALPHANUM	8			

LENGTH=n
or
BYTES=n

The LENGTH or BYTES operand specifies the length of a field as it is to be stored in the segment. The maximum value that can be specified for this operand is 255 bytes. The field length is stored in the Segment Layout Rule. The operand is also used to generate the Key Selection screen for data base segment fields and the Input Transaction Rule and Segment Display screen for both data base and pseudo segment fields. If the SLENGTH operand is not specified, the display lengths will be expanded by the Rules Generator on the basis of storage length whenever the field data requires conversion for display.

A DBD sequence field can be divided into more than one FIELD statement for display flexibility. The total length of these key fields must equal the DBD sequence field length. The maximum length DBD sequence field that can be processed by IMSADF II is 70 bytes. The LENGTH or BYTES operand is required for all fields in DBS, PS, and MAP segment definitions. The LENGTH or BYTES operand is not required for TEXT or KWNAME fields in output segment definitions. If one of these operands is not specified for TEXT fields, the text string length including blanks will be used. If it is not specified for KWNAME fields, the length of the keyword area in storage will be used.

For DB2 TYPE=VARCHAR, the LENGTH is the maximum length of the data area and does not include the halfword length. This halfword precedes the data area, dynamically specifying the data length.

The range of valid lengths by field type are:

TYPE	LENGTH
Alphabetic	1-255
Alphanumeric	1-255
Binary	2 or 4
Bit	1
Date	6
DBCS	2-256 (must be even number of characters)
Decimal	1-15
Float	8
Hexadecimal	1-255
Mixed	4
Numeric	1-255 (1-15 if used for Audit arithmetic)
Packed Decimal	1-8
Varchar	1-253

[MODE=n] (Conversational processing mode)

The MODE operand specifies the mode this field will be assigned in the Input Transaction Rule. The MODE operand applies only to fields specified with DISPLAY=YES. The segment display screen attributes will be dynamically set according to the mode in the Input Transaction Rule. Note that fields in the target segment for an Input Transaction Rule have a default of DISPLAY=YES. The modes that can be specified are:

4 (Modifiable, modifiable on transaction mode 6)
5 (Modifiable)
6 (Non-Modifiable)
7 (Non-Display, Modifiable)

When processing in transaction mode 6 and a field is defined as MODE=5, then non-key fields are protected and key fields are unprotected.

Defaults are set depending on whether a segment defined in an Input Transaction Rule is a target segment or a non-target segment. This is described under MODE for the SEGMENT statement. This operand overrides the MODE operand specified in the SEGMENT statement.

[MODE=n] (Nonconversational and Batch processing mode)

The MODE operand specifies whether or not the field is a net change field. A value of 8 specifies that data in the input Transaction Message is to be algebraically added in either the data base or Pseudo Segment data area. A value of 0 specifies that non-blank data is mapped to the field in the segment area and that the changed flag is set on in the associated Segment Layout Rule. The default value is 0.

The mode specified on the Nonconversational SIMAGE can be 4 through 7 with the same meaning as above, with the exception of modes 4 and 5. Mode 4 specifies a field is modifiable and the premodified data tag is off in the generated screen source. Mode 5 specifies a field is modifiable and the premodified data tag is on. To use the Auditor to set attributes for a field, a field must be identified as 4 or 5. If the transaction mode is 6, a mode 4 field must be specified to invoke the Auditor for process audits.

MSG= { **NO** | **YES** }

The MSG operand sets the MSG operand in the Segment Layout Rule. The MSG operand specifies whether or not an information message should be sent automatically to a Project/Group or user. A field is a candidate for auditing through the Message leg of the Audit Data Base if MSG=YES and the field has been changed or is specified KEY=YES. The default value is NO.

[NAME= {fieldname }]
[{(xxxxxxxx,SEQ,...)}]

The NAME operand refers to the name of the DBD sequence or non-sequence field name that is known to IMS/VS. The name is from one to eight characters in length and is placed in the Segment Handler Rule SSA list for DL/I calls. The primary use for this operand is to build SSAs using the DBD sequence field that is specified as KEY=YES on appropriate field statements. For multiple fields in a segment with KEY=YES specified, the KEYNAME operand of the SEGMENT statement must be used to override the NAME statement of the KEY=YES fields. The NAME operand can also be used to specify the DBD non-sequence field for a secondary compare, which is a Segment Handler option that is used only for Special Processing. The COMPARE and VALUE operands must also be specified to complete the Segment Handler Rule for a secondary compare DL/I call.

Field statements with the same NAME=DBNAME will be merged. When the DBD key has been subdivided into several ADFFIELD statements, the name parameter should not be used. The key name will be obtained from the KEYNAME operand of the segment statement. Duplicate names will cause the DBD key length to be set to the length of the last subdivided field.

If the segment does not have a unique sequence field defined in the DBD see "Processing Segments with Non-Unique Keys and No Keys" in Appendix D, "Additional Application Support Techniques" for a description of the supported functions and processing methods capable of handling the various situations.

[OFFSET=n]

This keyword is designed to allow a user to specify an offset from the beginning of the field specified on the REDEFINE keyword and will define where the sub-field will be placed. The OFFSET keyword will be used in conjunction with the REDEFINE keyword only. It will be ignored if REDEFINE is not specified. Reference REDEFINE keyword for an example of how OFFSET keyword may be used.

PAUDIT= {NO }
        {YES}

The PAUDIT operand specifies whether or not the field should be audited prior to displaying the segment display screen. This predisplay audit is accomplished by a special call to the Auditor from the transaction driver. Any fields in the Input Transaction Rule with PAUDIT=YES will be processed through the Auditor prior to the initial display of the screen for Conversational processing and prior to the mapping of input data for Nonconversational or batch processing. In either case, segments specified as loadable by (DBPATH) will be loaded into the SPA area prior to the preaudit call. The default is NO.

[POSITION=n]
or
[START=n]

The POSITION or START operand specifies the position of the first byte of a field in an application segment definition. POSITION=1 specifies the first byte of a segment. Specification of this operand results in the setting of positions in the Segment Layout Rules and Input Transaction Rule (for DB segments).

The POSITION or START operand is required only if fields are non-contiguous, i.e., there are spaces between field definitions or there are overlapping fields. If the POSITION or START operand is not specified, the Rules Generator will calculate position using the position and length of the preceding field. Bit fields will be positioned in the same byte if the offset is greater than the preceding bit field offset. If the offset is less or equal to the preceding offset, the position will be advanced one byte.

If POSITION=0 is specified for a field, the position of the preceding field will be used. See REDEFINE keyword for an example of the use of POS=0 parameter.

For DB2 the POSITION should not be specified for the base column definitions obtained from the DB2 catalog. The Rules Generator automatically calculates position using the position and length of preceding Columns. The Rules Generator does not do any boundary alignment for DB2 SMALLINT (halfword), INTEGER (fullword), DECIMAL (doubleword), FLOAT (doubleword) data types.

RDONLY= { **NO** | **YES** }

The RDONLY operand sets the RDONLY operand in a mapping segment definition. A value YES indicates that a user-routine can look at the data in the field, but that it cannot alter that data. The Data Mapper module will map data to the Special Processing Routine's or audit user exit's data area, but not back to the segment area in the SPA. The default value is NO.

[REDEFINE=fieldid]

This keyword allows a user to specify a field id where a sub-field's first field will start. The user may specify any field id within the current segment definition and should refer to a field which has been previously defined to the Rules Generator. This keyword specifies the field where start position is to begin in the redefinition. The first field of the redefine does not need an OFFSET= keyword. Subsequent sub-fields need OFFSET= to define the displacement to the start of that sub-field. If all the space of the original definition is not specified with OFFSET=, then the next field will start at the next available position following the original definition. If the user should enter an OFFSET= value which is outside the original field definition's length, an error message ADFG065 will be issued.

The following is a Rules Generator example:

```
SYSTEM SYSID=POS0,SOMTX=00,PGROUP=T5
SEGMENT ID=A1,PARENT=0,NAME=POSITION,LENGTH=99
FIELD   ID=KEYA,LENGTH=10,KEY=YES,POS=22
FIELD   ID=FLA1,LENGTH=03,POS=0
FIELD   ID=FLA2,LENGTH=7
FIELD   ID=FLA3,LENGTH=03,POS=0
FIELD   ID=FLA4,LENGTH=15,POS=35
FIELD   ID=FLA5,LENGTH=07,REDEFINE=FLA4
FIELD   ID=FLA6,LENGTH=05,REDEFINE=FLA4,OFFSET=7
FIELD   ID=FLA7,LENGTH=03,REDEFINE=FLA4,OFFSET=12
FIELD   ID=FLA8,LENGTH=04
GENERATE OPTIONS=(SEGL),SEGMENT=A1
```

This example causes the Segment Layout Rule to be built with the following positions: FLA1 will be at position 22 in the segment (POS=0 causes position of preceding field). FLA2 will be at position 32 in the segment. FLA3 will be at position 32 in the segment because of the POS=0 keyword. FLA4 will be at position 35 in the segment. The REDEFINE=FLA4 will cause FLA5 to start at position 35 for a length of 7. FLA6 will start at position 42 for a length of 5 and FLA7 will start at position 47 for a length 3. The field FLA8 with no REDEFINE and OFFSET will start at the next available position - 50 for a length of 4. Note that FLA5 does not have an OFFSET keyword and will default to OFFSET=0.

RELATED= { **NO** | **YES** }

The RELATED operand is specified if the field is both a non-key field and needed to clarify the key selection list on the Secondary Key Selection screen. A maximum of 72 bytes of data and column spacing, one screen line per segment occurrence, is available for both key and related field so that, usually, only a subset of a segment can be specified as related. The default value is NO.

REQUIRED= { **NO** / **YES** }

A YES value in the REQUIRED operand builds an FFRUL macro in the Input Transaction Rule for this field and sets REQ=Y. The value YES specifies that the contents of this field must be other than their initialized value. For Nonconversational and Batch processing the default value is YES for key fields and NO for non-key fields. For Conversational processing, the default is NO for all fields. Refer to the Auditing Section in Chapter 3, "Audit Language" for a discussion of initialized values.

[SCOL= { **n** / **(n,n)** } **]** **(1-160)**

The SCOL operand of the FIELD statement specifies the screen layout columns on which the SNAME literal and field data are to be displayed when explicit positioning is requested. See the SPOS operand on the GENERATE statement. If one column value is specified, then that value will be used for the field data column in the screen layout. In this case, the field literal will be right justified two columns preceding the data area. If two column values are specified and the literal and data areas are on the same row, then the screen format specification on the GENERATE statement will be used to provide justification or literal inserts of dashes between the SNAME characters and the field data area.

See the SPOS operand of the GENERATE statement for an example of a portion of a screen layout explicitly generated.

[SDECIMAL=n] **(1-13)**

The SDECIMAL operand specifies the number of digits to the right of the decimal point to be displayed on output. This operand is only applicable for TYPE=DEC or PD fields. The default value is the DECIMAL setting for the field.

SEGID=segid

The SEGID operand is used to build field names as part of a mapping rule or output format rule. The SEGID must refer to the two-character ID of a data base or pseudo segment. This operand is used when a mapping segment is being defined or an output segment field refers to a DB or pseudo segment field.

SIGN= { **YES** / **NO** }

The SIGN operand for a decimal or packed decimal field is used to select whether or not a sign indicator will be present in the stored format of the field. For packed decimal, SIGN=YES means that a positive number will be stored with a hexadecimal C in the low order 4 bits of the last byte. A negative number will have a hexadecimal D stored in the last byte. A SIGN=NO value means that a hexadecimal F will always be stored in the last byte. For zoned decimal, the hexadecimal C, D or F are stored in the high order four bits of the last byte. The TYPE operand may also be used to specify an unsigned decimal or packed decimal type, (TYPE=UDEC and TYPE=UPD), and will override the value specified for the SIGN operand. The default is SIGN=YES.

[SLENGTH=n]

The SLENGTH operand specifies the length of the field as it is to appear on the Segment Display screen. SLENGTH is valid only for conversational and nonconversational processing. If this operand is not specified for a field, the display length will be computed using the field length (LENGTH) as input. SLENGTH is valid only for conversational and nonconversational processing.

This computation is performed as follows:

Field Type	Display Length
ALPHA	LENGTH
ALPHANUM	LENGTH
BIN	8
BIT	1
DATE	LENGTH + 2
DBCS	LENGTH
DEC	LENGTH + 2
FLOAT	23
HEX	2 x LENGTH
MIXED	LENGTH
NUM	LENGTH
PD	2 x LENGTH + 1
UPD	2 x LENGTH +1
UDEC	LENGTH + 2
VARCHAR	LENGTH

[SNAME='1 to 30 characters']

The SNAME operand is used to build the literal for the field name in the Segment Display screen if DISPLAY=YES. If KEY=YES, SNAME is used to build the literal for the field name in the Primary Key Selection screen. When SKLEFT and SKRIGHT operands are not specified for a segment, SNAME also is used to build secondary key header text in the KEY Selection Rule for fields specified with KEY=YES or RELATED=YES. If space is not available to contain the header text on one line, each literal specified by an SNAME operand will be split into 2 header lines. The default value is one blank.

SQLISRT= {YES | NO} **(DB2 only)**

SQLISRT=NO identifies a Column that is not eligible for insert. This Column is not included in the VALUES clause of the SQL INSERT statement in the Table Handler Rule.

[SQLNAME='1 to 50 characters'] **(DB2 only)**

The SQLNAME operand identifies the DB2 column name as it is specified in the DB2 catalog. The SQLNAME will be the unqualified form (1 to 18 characters), or the table qualified from (1 to 46 characters).

The SQLNAME operand may also identify a column name with a built-in function or arithmetic expression. The Rules Generator will use the values between single quotes in the column list to be SELECTed. If a built-in function or arithmetic expression is used the column must be marked as SQLUPD=NO and SQLISRT=NO.

Quotes are required for all forms of SQLNAME, except the unqualified format of the column name with no built-in function or expression.

Every column in a Table specification must have a SQLNAME operand since DB2 addresses at the column level.

SQLNULL= {NO | YES} **(DB2 only)**

Identifies those Columns that can be NULL. A value NO indicates the DB2 Table definition for this column specified NOT NULL. If the value YES is specified, the column can be NULL. SQLNULL=YES implies SQLIND=YES at the TABLE level has been specified.

A Column can be marked NULL in an UPDATE or INSERT function through the setting of the indicator variable. The indicator variable is set through a screen input convention or through audits.

To set a column NULL, an indicator variable must be used (SQLIND=YES at the TABLE level) and the column must specify SQLNULL=YES.

Note: Indicator variables are generated for all DB2 columns, even though only those specifying SQLNULL=YES are used.

SQLORD= { ASC / DESC } **(DB2 only)**

SQLORD indicates the order of column values for those columns marked KEY=YES. ASC means ascending values and DESC means descending values. SQLORD will be used to build the KEY Selection SQL statement WHERE clause. SQLORD is also used in the KEY Selection SQL statement ORDER BY clause to define the sorting sequence (ASC or DESC). SQLORD value will be used by IMSADF II key selection processing to set HI or LO values in key fields.

The standard key selection browse function will have an ORDER BY clause associated with the SQL SELECT statement. This ensures that the rows displayed on the key selection browse screen will be in an ordered sequence. Even though a UNIQUE INDEX should be defined for a TABLE the DB2 BIND process may choose to ignore it and search the table using a TABLE scan technique. This would cause the rows to be displayed in an arbitrary order. Specifying an ORDER BY clause may influence DB2 to use the INDEX to avoid the overhead of the sort associated with using an ORDER BY clause.

SQLUPD= { YES / NO } **(DB2 only)**

SQLUPD identifies those Columns that may not be updated and should not be incorporated in the SQL UPDATE statement in the Table Handler Rule (SET clause). For example, if a View Column is a virtual column (i.e. derived from a built-in function), it cannot be updated in the SQL UPDATE call. Default is YES, except for KEY=YES columns which default to NO and cannot be changed to YES. KEY=YES columns are never eligible for update.

Arithmetic expressions will not be allowed in the SET and VALUES clauses of the UPDATE and INSERT calls. These functions can be accomplished by current operations in the Audit Language.

[SROW= { n / (n,n) } **]** **(6-62)**

The SROW operand of the FIELD statement specifies the screen layout rows or line numbers on which the SNAME literal and field data are to be displayed when explicit positioning is requested. See the SPOS operand on the GENERATE statement. If one row value is specified, then that value will be used for both the SNAME row and the field data row in the screen layout.

[START=n]
or
[POSITION=n]

The START or POSITION operand specifies the position of the first byte of a field in an application segment definition. START=1 specifies the first byte of a segment. Specification of this operand results in the setting of positions in the Segment Layout Rules and Input Transaction Rule (for DB segments).

The START or POSITION operand is required only if fields are non-contiguous, i.e., there are spaces between field definitions or there are overlapping fields. If the START or POSITION operand is not specified, the Rules Generator will calculate position using the position and length of the preceding field. Bit fields will be positioned in the same byte if the offset is greater than the preceding bit field offset. If the offset is less or equal to the preceding offset, the position will be advanced one byte.

See REDEFINE keyword for explanation of POS=0 parameter.

[TEXT='1 to 30 characters']

The TEXT operand is used to include a text string or IMS/VS transaction code in an output segment definition. The maximum

length of each text string specified is 30 characters. Consecutive FIELD statements with TEXT operands may be specified to provide any length text string. When used to specify an IMS/VS transaction code, either a trailing blank should be entered as part of the text string or a LENGTH operand must be specified to account for an additional character. A LENGTH operand is not required if the text string length is the exact length desired.

TYPE=	ALPHANUM	C	(alphanumeric characters)
	ALPHA	A	(alpha characters)
	BIN	I	(binary number)
	BIT	B	(bit data)
	DATE	DA or D	(data stored as yymmdd displayed with format defined at installation)
	DBCS	DB	(double byte character set)
	DEC	DE or Z	(zoned decimal)
	FLOAT	F	(floating point)
	HEX	H or X	(hexadecimal representation)
	MIXED	M	(mixed EBCDIC and DBCS)
	NUM	N	(numeric characters)
	PD	P	(packed decimal)
	UDEC	UD or UZ	(unsigned zoned decimal)
	UPD	UP	(unsigned packed decimal)
	VARCHAR	V	(variable character)

The TYPE operand specifies either how a field is represented in a data base segment, how a pseudo segment is represented in the data storage area, or how a Special Processing Routine data area field is to be mapped.

The default value is ALPHANUM. For Standard Processing, fields with KEY=YES can not have the value BIT.

[VALUE='1 to 30 characters']

The VALUE operand specifies the compare value that will be added to the SSA following the Boolean operator in the COMPARE operand. The length of the compare value in the Segment Handler Rule SSA is equal to the field length in the LENGTH operand. The Rules Generator will truncate or pad VALUE text to accommodate the field length. Refer to the COMPARE description for situations in which the VALUE operand is specified. The default value is blanks. A maximum of 30 characters can be entered. Non-alphanumeric and non-alpha field type values are not converted.

TABULAR INPUT FOR FIELD DEFINITIONS

Field definitions can be entered either in keyword format or by a tabular input form. A colon character (:) appearing in column one of an input statement designates that statement as a tabular form of the field statement. The operands that may be entered on this form, for data base and pseudo segment definitions, are ID, START or POSITION, TYPE, LENGTH or BYTES, KSELECT, ASTATUS, DMODE, SLENGTH, and SNAME or KEYBLD/COFIELD. The tabular field statement may be used as an original field definition or as a field override. If used as a field override, the ID operand must contain a segment qualifier (ffff.ss) or the DBD name. If the DBD name is entered, the START operand may contain a field ID in place of the start position. This technique may be used to override DBD statements.

The LENGTH or BYTES and SLENGTH operands have been extended to include the decimal factor as a second value for decimal and packed decimal field types. In that case the two values must be separated by a comma. For example, an operand value (6,3) indicates a field length of 6 with digits following the decimal point. For BIT field types, the second value is used as the bit offset.

Fields that are used in the key selection phase as key build fields or have a cofield (KSELECT=KB or KC or RC), must have the referenced field name as the first non-blank characters of the SNAME operand. An SNAME literal may follow the reference field on the same tabular statement. In this case, one blank character following the reference field is used

as a delimiter. A full 30-character SNAME for the field may be entered on a tabular field override statement (see field FLD4 in the example below).

A segment specification consists of any combination of tabular and keyword field statements. Either form may be used to override an existing field definition. However, field operands COLUMN, RDONLY, COMPARE, VALUE, SROW, SCOL, FLDPOS, ILENGTH and ITYPE are not in the tabular form and must be specified by keywords.

The following is an example of a data base segment definition using tabular field definitions as the primary input method:

```
************************
 SEGMENT ID=GF,LEN=68,PARENT=0,NAME=MFGFAR01,KEYNAME=MFGFGRFL,
         KASCEND=NO,SKLEFT='KEYS AND RELATED DATA',SKSEGS=18
************************
* COLUMNS  1     1  1     2  2        3  3     4                            7
*2         1     6  9     4  7        5  8     3                            2
* ID.XX   |STRT|TY|BYTE|KS|ASTATUS|DM|SLEN|    SNAME  OR  FIELD.SEGID
: GRFL    |    |C :  8 :K :  R     |Y5|    :SYSTEM ID/AUDIT GROUP
: FLID    |    |C : 28 :K :  R     |Y5|    :FIELD NAME (SSXXFFFF)
: FLD1    |    |C :  2 :K :YP      |N | 4  :KEY FIELD/DISPLAY
: FLD2    |    |DA:  6 :KA:YAP     | Y|9   :DISPLAY/PRIMARY/AUDIT/SEC AUD
: FLD3    |    |I :  2 :KB:YAPFMRC| N|    :GRFL.GF KEY BUILD/NODISP-AUD
: FLD4    |    |N :  2 :KC:Y       |4 |   5:PSF1.PS
: FLD4    |    |  :    :  :Y       |  |    :KEY FIELD/COFIELD INPUT TO 72
: FLD5    |    |P :  2 :KN:PAMC    |3 |3,1 :NO-DISPLAY/PRIMARY AUDIT
: FLD6    |    |X :  2 :KP:  M     |6 | 5  :DISPLAY/PRIMARY AUDIT
: FLD7    |    |C :  3 :KS:M       |  |    :DISPLAY/SECONDARY AUDIT
: FLD8    |    |DE: 6,2: R: AF     |7 |    :RELATED FIELD LITERAL
: FLD9    |    |UP:  6 :RS:  PFA   |  |16,5:RELATED DISPLAY/SECONDARY AUD
: FLD0    |    |UD:  2 :RC:  AF    | 4|    :PSF2.PS
: FLD0    |    |  :    :  :   F    |  |    :RELATED/COFIELD/SEC AUD
:DBDNAME  | 40 |A :  2 :N :YP      |Y | 3  :NON-KEY/NOT-RELATED/NO-DISP
:DBDNAME  |FLDA|  :    :  :        |  |    :DBDNAME OVERIDE
: FLDB    |    |B : 1,4:N :        |Y | 1,3:NON-KEY/NOT-RELATED/NO-DISP
FIELD ID=FLD6,FLDPOS=32,ITYPE=C,ILENGTH=4
GENERATE OPT=SEGL,SEG=GF
```

The tabular input for field definitions that represent a mapping or output segment layout is a subset of the tabular input form for data base and pseudo segments. Operands that may be entered for mapping or output segment definitions are ID, START or POSITION, TYPE, LENGTH or BYTES and TEXT. The ID operand must be a segment qualified field ID(ffff.ss), a DBD name, a KWNAME keyword (see KWNAME operand for the list of keywords), or the &TEXT keyword. Keyword names must be specified with an ampersand character (&) immediately preceding the name, such as &SPATRAN. A unique set of characters must be entered for for a keyword up to a maximum of seven (the ID field is eight columns wide). The &TEXT keyword is used to indicate that columns 43-72 are a text string for the output format layout. A field length entry is required on &TEXT fields if trailing blanks must be preserved.

The following is an example of an output segment definition. A mapping segment definition would only contain the field references to data base and pseudo segment fields (ffff.ss) as an ID operand value.

```
************************
 SEGMENT ID=M1,TYPE=OUT
************************
* COLUMNS 1     1  1     2                  4                        7
*2        1     6  9    *4                 *3                        2
* ID.XX  |STRT|TY|BYTE|  LEAVE BLANK   |     TEXT OUTPUT
: &TEXT  |    |C :  9 |                :TRANCOD1
:FLID.GF |    |C : 28 |                :
:&COSEGR |    |  :    |                :
:FLD9.GF |    |C :16,4|                :
: &TEXT                                :END OF OUTPUT
```

If field overrides are to immediately follow MAP (or OUTPUT) segment FIELD definitions, then a null GENERATE statement is required to separate the MAP (or OUTPUT) fields from the field overrides. A null GENERATE statement contains no options or parameters. When the Rules

Generator encounters a null GENERATE statement, previous segment pointers are cleared.

```
SEGMENT   ID=M2,TYPE=MAP
FIELD     ID=FLID.GF,BYTES=28          /* MAP field      */
GENERATE                               /* Null GENERATE  */
FIELD     ID=PART.PA,AFA=YES           /* FIELD override */
```

GENERATE STATEMENT

The GENERATE statement is used to generate static rules for conversational, nonconversational, and batch processing, and screens for conversational and nonconversational processing. The GENERATE statement is also used to generate executable transaction driver load modules for conversational, nonconversational, or batch processing. For segment related generation, the GENERATE statement either must be positioned after the SEGMENT and FIELD statements that require rule or screen generation or it must refer to some segment that was defined previously. A subset of the operands from the SYSTEM statement can be specifically overridden by the GENERATE statement. The overrides apply only to current rule and screen generation.

Generation requests are broken into five major categories:

1. Application System related rules for Conversational Processing
   Sign-on Screen and Primary
   Option Menu Rule

2. Segment related rules
   Segment Layout Rule,
   Segment Handler Rule

3. Transaction related rules and screens
   Input Transaction Rule,
   Primary Key Selection screen, Segment Display screen, Text Utility screen

4. Secondary Option Menu Rule
   Conversational Processing

5. Transaction driver link-edits, preload option and Composite Rules Load Module
   Transaction drivers for Conversational,
   Nonconversational, Batch Processing, Preload Rule and
   Composite Rules Load Module

An OPTIONS operand is used to specify which rule(s) and screen(s) are to be generated. A separate GENERATE statement must be used for each OPTIONS operand. Some rules may be generated in conjunction with certain other rules, using a single OPTIONS operand, while certain other rules should never be generated together. In general, the rules that belong to the same category may be generated together.

With each rule are associated operands that define that rule. The following are nine separate operand summary figures for the GENERATE statement showing the various generate options with their associated operands. Figure 2-8 on page 2-52 lists operands for the application system related Sign-on screen and the Primary Option Menu Rule. Figure 2-9 on page 2-53 lists operands for segment related rules. Figure 2-10 on page 2-54 lists operands for transaction related rules and screens. Figure 2-11 on page 2-57 lists operands for Secondary Transactions. Figure 2-12 on page 2-58 lists operands for the Secondary Option Menu. Figure 2-13 on page 2-58 consists of operands for IMSADF II Extract. Figure 2-14 on page 2-59 lists operands for requesting link-edits. Figure 2-15 on page 2-61 summarizes the operands for generation of a Preload Rule. Finally, Figure 2-16 on page 2-62 summarizes the operands for generation of a Composite Rules Load Module.

The set of rules required for transactions in each of the three processing modes (Conversational, Nonconversational, Batch) vary somewhat, and some operands are unique to each processing mode. Those operands which are applicable to a specific processing mode, and those which are applicable to special processing or text utility processing,

are noted in the summary tables. Following the summary figures is a discussion of the generation options for each processing mode and for requesting link-edits.

The generated rules and screens are stored in a PDS rules library and may be used by all transactions within the same application system ID. Thus, new transactions may be built using existing rules generated by previous transaction definitions. However, newly generated rules and screens with the same SEGMENT IDs or TRXIDs will replace existing ones.

OPERANDS	VALUES	DESCRIPTION
OPTIONS= OP	({SIGN,POM}) CVSYS	request for Sign-on screen request for Primary Option Menu Rule equivalent to (SIGN,POM)
Sign-on screen Request		
DEVCHRS= DEVC	(nn,nn,...)	device characteristics
DEVNAME= DEVN	([n [,An][,...]) An	device for this Sign-on screen
DEVTYPE= DEVT	([n,...])	device models for this Sign-on screen
[SHEADING= SHE	'1 to 54 chars']	heading for Sign-on screen
SIGNON= SI	YES NO RACF	request for Sign-on screen
SOIMAGE= SOI	NO YES	screen image used for Sign-on screen
Primary Option Menu Request		
LINKLIB= LINKL	SYSLMOD xxxxxxxx	library for Primary Option Menu Rule
POMENU= POME	ALL ({A,B,C,D,F,H,I})	request for Primary Option Menu Rule

Figure 2-8. GENERATE Statement Summary (System Related)

OPERANDS	VALUES	DESCRIPTION
OPTIONS= OP	({SEGL,SEGH}) SGALL	request for Segment Layout Rule for all segment types request for Segment Handler Rule for all data base segments equivalent to (SEGL,SEGH) SEGMENTS operand optional
[ALIAS= ALI	(sysid,...)]	create SEGL and SEGH Rules for different SYSID's in same Rules Generation Run.
LINKLIB= LINKL	SYSLMOD xxxxxxxx	data set for Rules Generator output
[SEGMENTS= SEGM	(segid,...)]	segments for which rules are generated
Segment Handler Rule Only		
MSGDB= MSGD	0 n	message number when data base read fails
MSGNF= MSGN	0 n	message number when data base segment not found

Figure 2-9. GENERATE Statement Summary (Segment Related)

OPERANDS	VALUES	DESCRIPTION
Request for Generation		
OPTIONS= OP	({INTR,KEYD,SEGD}) CVALL ({TEXT,KEYD}) TUALL TPALL TPIT BAIT	Conversational Processing Rule and screens Conversational Processing Rule and screen for text segment Nonconversational Processing Rules and screen Batch Processing Rule
Transaction Identification		
AGROUP= AG	YYYY xxxx	audit group ID
ASMREQ= ASMR	NO YES	invoke Assembler
[DBPATH= DB	(segid,...)]	data base path definition
DLIEXIT= DLI	NO YES	invoke a DL/I Exit during this Input Transaction Rule.
KANAME= KAN	ALT * STD	specify whether key audits are to be accessed under the root key.
LINKLIB= LINKL	SYSLMOD xxxxxxxx	data set for Rules Generator output
LINKREQ= LINKR	NO YES	specify a link-edit on all rules generated to this point.
LRULE= LR	NO ALT YES	request for retrieving audit static rules
TRXID= TRXI	trxid	two-character transaction ID
[TSEGS= TS	(segid,...)]	pseudo segments or data base segments not in DBPATH
USRLANG= USR	x (A-Z,0-9) *	language to be associated with SYSID
Transaction Exit Specifications		
SPECIAL= SPE	NO YES	special processing Input Transaction Rule
LANGUAGE= LA	COBOL ASMINT ASMSTD PL/I	special processing language

* see parameter description for default value

Figure 2-10 (Part 1 of 4). GENERATE Statement Summary (Transaction Related)

OPERANDS	VALUES	DESCRIPTION
BYPASS= BY	{NO YES}	time of special processing call (Conversational and Batch only)
DYNLOAD=	{NO YES}	Special Processing Routine dynamically loaded
[STX= ST	({MFS TRX},segid[,OK][,ER] [,{0 n}])]	Secondary Transaction
Delete and Insert Eligibility		
[DLET= DL	(segid,...)]	segments with delete eligibility
[ISRT= IS	(segid,...)]	segments with insert eligibility
For Conversational and Nonconversational		
[CURSOR= CU	({fieldid field.segid adfname} ,...)]	cursor position for Segment Display screen
DAMSG= DAM	{NO YES}	request for delete or insert messages
DEVCHRS= DEVC	(nn,nn,...)	device characteristics
DEVNAME= DEVN	([n [,An][,...]) An	IMS/VS identification for display devices
DEVTYPE= DEVT	([n,...])	screen size(s) for this transaction
[IMAGE= IM	xxxxxxxx]	screen to be included
PFKDATA= PFKD	{NO YES}	specifies that PFK literals are to be used as input transaction data.
PFKLIT= PFKL	('22 chars') (10='19 char')	literal to be passed to IMS/VS or passed to IMSADF II when program function key is entered.
PFKNUMB= PFKN	n	number of PFK literals to generate if PFKDATA = yes and PFKLIT not specified

Figure 2-10 (Part 2 of 4). GENERATE Statement Summary (Transaction Related)

OPERANDS	VALUES	DESCRIPTION
For Conversational Processing		
ADDMODE= AD	{NO YES}	specifies transaction mode to follow add
COMMLEN= COM	{0 n}	length of user-maintained communication area
[DATACOMP= DAT	(segid,...)]	segments compared before update
DISPNAME= DI	{MO PG}	Segment Display screen name
DKEY= DKE	{YES NO}	display key on Segment Display screen
DTRAN= DTR	{YES NO}	display transaction code on Segment Display screen
DTWINC= DTW	{YES NO}	flag to Auditor for validity check of keys of twin segments to be inserted.
KEYSL= KE	{NO YES} *	key selection required
MAXKEY= MAX	{50 n}	screen key length
[PGROUP= PGR	xx]	project group
SFORMAT= SF	{DASH LEFT RIGHT}	screen format for literals
[SHEADING= SHE	'1 to 54 chars']	screen heading
SOMTX= SOMT	tx	Secondary Option Menu cluster code
SPOS= SPO	{AUTO ROWCOL SIMAGE}*	type of screen generation
TRXNAME= TRXN	'1 to 30 chars'	transaction name for screen generation
[TWINS= TW	(segid,...)]	list of twin segments
[VACANT= VA	(segid,...)]	list of twin segments for insertion

* see parameter description for default value

Figure 2-30 (Part 3 of 4). GENERATE Statement Summary (Transaction Related)

OPERANDS	VALUES	DESCRIPTION
For Nonconversational Processing		
MODNAME= MODN	xxxxxxxx	transaction name known to user
[ORID= OR	segid]	Output Format Rule for MFS output message
For Batch Processing		
CNT= CN	{ 1 } { n }	number of input cards for transaction
EOF= EO	{ NO } { YES }	end of file operand

Figure 2-10 (Part 4 of 4). GENERATE Statement Summary (Transaction Related)

OPERANDS	VALUES	DESCRIPTION
Request for Generation		
OPTIONS= OP	OMFS	Output Format Rule and screen source for secondary transactions
[ORID= ORI	xx]	Output Format Rule I/O for secondary transactions
[PRINTER= PRI	{ 1 } { 2 }] { 3 }	direct secondary transaction output to a printer
SPOS=	SIMAGE	screen image

Figure 2-11. GENERATE Statement Summary (Secondary Transactions)

OPERANDS	VALUES	DESCRIPTION
For Conversational Processing Only		
OPTIONS= OP	SOM	request for SOMR generation
[DELETE= DEL	trxid,...)]	request for transaction deletion from SOMR
LINKLIB= LINKL	{ SYSLMOD xxxxxxxx }	data set for Rules Generator output
SOMEND= SOME	{ YES NO }	request for SOMR deck
UPDATE= UP	{ NO YES } *	request for rule update
OPTIONS= OP	CSOM	mark specified transactions in SOMR as having a Composite Rules Load Module
COMP=	{ NO YES }	Composite Rules Load Module indicator
CTRX= CT	(trxid,...)	transactions to be placed in Composite Rules Load Module

* see parameter description for default value

Figure 2-12. GENERATE Statement Summary (Secondary Option Menu Rules)

OPERANDS	VALUES	DESCRIPTION
Request for Extract		
OPTIONS= OP	ADFX	invoke IMSADF II Extract Processing
[DDAFX= DD	DDNAME]	data set for IMSADF II Extract
DECKS=	{ NO YES }	request for rules source deck
[SEGMENTS= SEGM	(segid,...)]	segment list for IMSADF II Extract to process

Figure 2-13. GENERATE Statement Summary (IMSADF II Extract)

OPERANDS	VALUES	DESCRIPTION
Request for Link-edit		
OPTIONS= OP	BDLE NCLE SPLE STLE TPLE	Batch Transaction Driver Nonconversational Standard Process Conversational Special Processing Conversational Standard Processing Nonconversational Special Process
[AEXIT= AE	xxxxxxxx]	audit exit module name
ALANG= ALA	COBOL ASMINT ASMSTD PL/I	audit exit language
[DEXIT= DEX	xxxxxxxx]	DL/I exit module name
DLANG= DLA	COBOL ASMINT ASMSTD PL/I	DL/I exit language
DYNLOAD= DYN	NO YES	special processing dynamic load
LINKLIB= LINKL	SYSLMOD xxxxxxxx	data set for link-edit output
PGMID= PGM	xx	program ID
Special Processing Link-edit		
LANGUAGE= LA	COBOL ASMINT ASMSTD PL/I	Special Processing Routine language
[MAPTABLE= MAP	(segid,...)]	Mapping Rules
[SHTABLE= SHT	(segid,...)]	Segment Handler Rules

Figure 2-14 (Part 1 of 2). GENERATE Statement Summary (Link-edit)

OPERANDS	VALUES	DESCRIPTION
Batch Transaction Driver Link-edit		
CHKPT= CH	{NO YES}	request for IMS/VS checkpoint
FREQ= FR	{0 n}	frequency of checkpoints
[ITTABLE= IT	(trxid,...)]	Input Transaction Rules
LANGUAGE= LA	{COBOL ASMINT ASMSTD PL/I}	Special Processing Routine language
[PHEADING= PH	'1 to 60 chars']	printer heading
SIGNON= SI	{NO YES}	request for Sign-on security
[SLRTABLE= SL	(segid,...)]	Segment Layout Rules for pseudo segments
[SPNAMES= SPN	(sprname,...)]	special processing names
[SPTABLE= SPT	(trxid,...)]	special processing Input Transaction Rules
[WTOMSG= WTOM	'1 to 60 chars']	WTO message
[WTORMSG= WTOR	'1 to 60 chars']	WTOR message

Figure 2-14 (Part 2 of 2). GENERATE Statement Summary (Link-edit)

OPERANDS	VALUES	DESCRIPTION
OPTIONS= OP	PREL	request for Preload Rule
[ASRTABLE= ASR	(auditgrp,...)]	preload Audit Static Rules
[DASRTAB= DASIT	(auditgrp,...)]	delete existing preloaded audit group
[DELETE= DEL	(segid,...trxid,...)]	delete existing preloaded rules
[ITTABLE= IT	(trxid,...)]	preload conversational Input Transaction Rules
[MAPTABLE= MAP	(segid,...)]	preload Mapping Rules
[OFRTABLE= OF	(segid,...)]	preload Output Format Rules
PREPOM= PREP	{NO YES}	preload Primary Option Menu Rule
PRESOM= PRES	{NO YES}	preload Secondary Option Menu Rule
[SHTABLE= SHT	(segid,...)]	preload Segment Handler Rules
[SLRTABLE= SL	(segid,...)]	preload Segment Layout Rules
[SPTABLE= SPT	(trxid,...)]	preload special processing Input Transaction Rules
[TPITTABL= TP	(trxid,...)]	preload nonconversational Input Transaction Rules
UPDATE= UP	{NO YES} *	update Preload Rule or not

* see parameter description for default value

Figure 2-15. GENERATE Statement Summary (Preload Rule)

OPERANDS	VALUES	DESCRIPTION
OPTIONS= OP	{CTLE} {NTLE}	Conv. Composite Rules Load Module Nonconv. Composite Rules Load Module
[ASRTABLE= ASR	(auditgrp,...)]	Composite Audit Static Rules
[ITTABLE= IT	(trxid,...)]	Composite conversational Input Transaction Rules
[MAPTABLE= MAP	(segid,...)]	Composite Mapping Rules
[OFRTABLE= OF	(segid,...)]	Composite Output Format Rules
[SHTABLE= SHT	(segid,...)]	Composite Segment Handler Rules
[SLRTABLE= SL	(segid,...)]	Composite Segment Layout Rules

Figure 2-16. GENERATE Statement Summary (Composite Rules Load Module)

GENERATE STATEMENT OPTIONS OPERAND

APPLICATION SYSTEM RELATED OPTIONS FOR CONVERSATIONAL PROCESSING

```
OPTIONS= {({SIGN,POM})}
         {CVSYS       }
```

The OPTIONS=SIGN operand value is used to specify that a Sign-on screen is to be generated for the current application system ID. Associated operands SHEADING, DEVNAME, DEVCHRS, and DEVTYPE may be specified with this option.

If the Sign-on screen has already been generated on a previous SYSTEM or GENERATE statement, additional requests for sign-on will result in error messages if screen image was used in either place. If no duplicate request used screen image, the subsequent requests are ignored.

The OPTIONS=POM operand value is used to specify that a Primary Option Menu Rule is to be generated for the current application system ID. The associated POMENU operand may be specified with the OPTIONS=POM value if all Primary Option Menu options are not required (i.e., POMENU=(A,...)).

If the POM was already generated on a previous SYSTEM or GENERATE statement, it will be ignored here and it will not be regenerated.

The OPTIONS=CVSYS operand value is used to specify that both a Sign-on screen and a Primary Option Menu Rule are to be generated for the current application system ID. The OPTIONS=CVSYS operand value is equivalent to specifying OPTIONS=(SIGN,POM). The POMENU operand is set to ALL as a default and may be entered on the GENERATE statement to subset the options.

SEGMENT RELATED OPTIONS

```
OPTIONS= {({SEGL,SEGH})}
         {SGALL        }
```

The OPTIONS=SEGL operand value is used to specify the generation of Segment Layout Rules for data base and pseudo segment definitions, Mapping Layout Rules for mapping segments and Output Format Rules for output segments. The associated operand SEGMENTS must also be entered to reference those segments for which segment layouts are to be generated.

The OPTIONS=SEGH operand value is used to specify the generation of Segment Handler Rules for data base segments. The associated operand SEGMENTS must also be entered to reference those segments for which Segment Handler Rules are to be generated.

Note that when there are multiple eligible segments the Segment Handler Rule operands MSGNF and MSGDB are repeated for each segment. These operands are commonly used when a transaction has one target segment and a very specific diagnostic is required. If the MSGNF and MSGDB operands are not specified, IMSADF II drivers will generate an appropriate message for any particular segment with a retrieval error.

The OPTIONS=SGALL operand value may be used to request segment related rule generation for all segments defined prior to this GENERATE statement. If the SEGMENTS operand is specified, however, only those segments referenced are eligible for this rule generation. When OPTIONS=SGALL is specified the appropriate segment related rules are generated for each segment depending on the type of segment definition as follows:

Segment Type	Rules Generated
DBS	Segment Layout Rule and Segment Handler Rule
PS	Segment Layout Rule
MAP	Mapping Segment Layout Rule
OUT	Output Format Rule

Once a segment has been processed using the OPTIONS=SGALL operand it is no longer eligible for processing unless specifically referenced by the SEGMENTS operand or the LINKLIB operand is also entered.

OPTIONS=TABL or SEGL (DB2 only)

A Table Layout Rule is required of each Table/View to be processed by IMSADF II. The GENERATE option TABL/SEGL will build the Table Layout Rule control block describing the data type, length, offset, etc. of each column in the DB2 Table View. This information is utilized at execution time by the transaction drivers to interpret the data returned and updated through DB2 SQL.

Since DB2 addresses data at the column level, the Table Layout Rule will match the Assembler DSECTs generated in the Table Handler Rule. Neither the Table Layout Rule or the Assembler DSECTs generated in the Table Handler Rule will be boundary aligned.

The Format of the GENERATE statement is:

```
GENERATE OPTIONS=TABL,TABLES=(id,id,...id)
```

OPTIONS=TABH (DB2 only)

A Table Handler Rule is required for each Table/View ID to be processed by the IMSADF II transaction drivers through standard processing functions, High Level Audit Language operations or the SQLHNDLR Exit function. The OPTION 'TABH' will build an Assembler program, containing IMSADF II standard static SQL calls and user specified SQL calls. The standard calls are SELECT for single row inquiry, SELECT for multiple key selection row browse, UPDATE, INSERT, CURSOR SELECT, CURSOR UPDATE, and CURSOR DELETE. This program will be built by the Rules Generator and processed by the DB2 pre-compiler, the Assembler, and the Linkage Editor.

The Assembler is invoked with the Assembler (NOALIGN) parameter in effect to ensure that Assembler DSECTs are assembled without boundary alignment. This ensures that the Assembler DSECTs match the Table Layout Rule.

The Format of the GENERATE statement is:

```
GENERATE OPTIONS=TABH,TABLES=(id,id,...id),SQLCALL=,SQLUSER=
* 1-8     10-17    19-71
label     sqlfunc WHERE clause
*                16 to 71 continuation
&SQLENDS
```

TRANSACTION RELATED OPTIONS

OPTIONS= { **({INTR,KEYD,SEGD})** / **CVALL** / **({TEXT,KEYD})** / **TUALL** } **(Conversational)**

The OPTIONS=INTR operand value is used to specify the generation of an Input Transaction Rule for Conversational processing. The OPTIONS=KEYD operand value is used to specify the generation of a Primary Key Selection screen. The OPTIONS=SEGD operand value is used to specify the generation of a transaction related or Segment Display screen. If OPTIONS=SEGD or OPTIONS=KEYD and a cofield is specified for display, OPTIONS=INTR must also be specified for that transaction. The INTR, KEYD, and SEGD option requests relate to a specific transaction and must be identified using the associated operands TRXID and TRXNAME. The Segment Display screen request may have the associated operands SPOS and IMAGE specified for requesting automatic or image layout generation options.

The OPTIONS=CVALL operand value specifies that a Conversational Input Transaction Rule and the associated screens are to be

generated, (i.e., OPTIONS=CVALL is equivalent to OPTIONS=(INTR,KEYD,SEGD)).

The OPTIONS=TEXT operand value specifies that a Conversational Input Transaction Rule for a text segment and a Text Utility screen are to be generated for the applicable data base path target segment. This option must only be used for segments with key lengths less than or equal to eight bytes in length and having a total segment length of less than or equal to 78 bytes. The key must be the first field in the segment and all data in the segment must be stored in a displayable format (i.e., Alpha, Alphanum or Num). The OPTIONS=TUALL operand value specifies that a Conversational Input Transaction Rule for a text segment and the Text Utility screen is to be generated, (i.e., OPTIONS=TUALL is equivalent to OPTIONS=(TEXT,KEYD)).

OPTIONS= {TPIT | TPALL} **(Nonconversational)**

The TPIT option will create an Input Transaction Rule for the transaction ID specified by the TRXID operand.

If a nonconversational transaction is to be entered via a formatted screen for response to the entering terminal, it may be generated by specifying the TPALL operand value. The TPALL option causes a TPIT rule and a Segment Layout rule for an Output Format rule to be generated in addition to the Segment Display Screen necessary for this type of Nonconversational transaction. Also, an error display screen is associated with the Segment Display Screen. An Output Format Rule matching the display screen MOD is automatically generated. The FLDPOS operand value of the FIELD statement used in the generated TPIT rule is automatically created for each unprotected field in the screen image. A screen image input definition (SPOS=SIMAGE) must be entered to invoke the TPALL option. (See section "Screen Image Input" on page 2-101.)

OPTIONS=BAIT **(Batch)**

The BAIT option will create an Input Transaction Rule for the transaction ID specified by the TRXID operand.

Output Format Rule and Screen Source for Secondary Transactions

OPTIONS=OMFS

- The OMFS option is to be used in conjunction with the ORID= parameter, the SPOS=SIMAGE parameter, and, optionally, the PRINTER= parameter. When these specifications are made, the Rules Generator will build an output format rule and screen source from the screen image following the generate statement. The two-character ID used on the ORID will be used by the Rules Generator in naming the Output Format Rule as well as the DOF and the MOD in the screen source. It will also be used by the user in a later STX operand to identify which output format rule is to be used. The following fields may be used on the screen image for output format rules for secondary transactions:

  1. Literals - Literal data is entered directly on the screen image statements at the required screen locations.

  2. Application fields - Application fields may be entered at the desired screen locations as &6ffff or &6ffff.ss where: ffff is a one to four character field ID of a field to be included. This ID must match the field ID of a previously specified field statement in one of the data base segments or pseudo segments defined in the Rules Generator input stream. ss specifies a two-character segment ID of the field to be included. This ID is used to specify the correct segment definitions in cases in cases where non-unique field names are present.

  3. Control fields and symbolic references - These fields are used the same as described in the procedure documented for screen image for segment display screens.

- Screen image input can be entered following the generate statement or can be included as a member of IMAGELIB.

- The PRINTER= parameter may be used to direct output to a printer. This specification tells the Rules Generator which device characteristics to use for building the screen source. The values allowed and line widths used are 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

  When the PRINTER= parameter is not used, default is the 3270 model 2 terminal.

- When a printer is specified, and the screen image follows the GENERATE statement, two card images are required for each printer layout line. The first 66 characters of the first card and the first xx characters of the second card are processed as image characters, where xx is 54 for the 3270 printers (printer=1 or 2) and xx=66 for the SCS1 printer (printer=3).

- When a printer is specified, and the screen image is to be included from the IMAGE= parameter, then the screen image should represent the exact output layout. (i.e., the two card image is invalid.)

- System fields cannot be used on screen image for secondary transactions. They will be ignored by the Rules Generator.

- Quotes cannot be used on screen image for secondary transactions.

- A literal cannot be continued across a printed line.

- Data cannot extend into column 120 for PRINTER = 1 or 2 and column 132 for PRINTER=3.

  When IMSADF II builds the DFLD's for the screen image, whether the SIMAGE is in IMAGELIB or not, it uses a blank as a delimiter. If one line of the screen image is coded with literals to the allowable maximum length, then the following SIMAGE line may not begin in column 1. If so, IMSADF II will treat this card as a continuation and the DFLD will show an overlap. The same applies in reverse. If a line of the SIMAGE is to begin in column 1, then the previous line in the screen image may not be filled with literals in the last column. (i.e., the last column must be blank.)

- Fields are placed into the output format rule in the order they are defined in the Rules Generator input stream, regardless of segment type.

- No output segment (TYPE=OUT) is defined in the Rules Generator input stream. The data used must be defined in data base or pseudo segments in the run. If fields in different segments have the same name and the user does not specify SEGID (ss above), the user is not assured of picking up the correct field.

- No DIF or MID is generated because the transaction is for output only.

SECONDARY OPTION MENU OPTION FOR CONVERSATIONAL PROCESSING

OPTIONS=SOM

The OPTIONS=SOM operand value specifies that a Conversational Secondary Option Menu is to be generated. All transaction IDs for which an Input Transaction Rule was generated will be included in the rule. OPTIONS=SOM without the UPDATE operand specified causes the existing Conversational SOMR to be updated with the transaction IDs and transaction names of all ITRs generated within the run. If a SOMR is not found, a new SOMR will be created. A STEPLIB DD statement pointing to the Rules Load library is required if new entries are added to an existing SOMR. If UPDATE=YES is explicitly specified and a SOMR cannot be found, the generation is terminated with an error condition. UPDATE=NO specifies that a new SOMR is

created with the transaction IDs generated in this run. All Standard, Text, and Special Processing transaction IDs are included in the SOMR. A maximum of 300 transactions is allowed for each SOM rule.

If a SYSID is specified on the SOM GENERATE, the Rules Generator will build a secondary option menu rule based on that SYSID. If multiple SOM GENERATEs are entered with different SYSIDs, the Rules Generator will build an SOM rule for each of the SYSIDs. If SOM GENERATE is coded, without a SYSID specified, secondary option menu rules will be processed for all SYSIDs when input transaction rules have been generated in this rules generation run.

OPTIONS=CSOM

The OPTIONS=CSOM operand specifies the transactions in the SOM rule to be marked as having Composite Rules Load Module. The format of this option is:

```
GENERATE OPTIONS=CSOM,COMP=value,CTRX=(trxid,trxid,trxid....)
                                        or
                                        ALL

where

      value is YES -- meaning set composite indicator on
                   or
               NO  -- (default) no composite module exists

      trxid is the TRXID(s) whose indicator(s) are to be set as COM
                   or
               ALL meaning all TRXIDS in this system are set as COM
```

The purpose of CSOM is to update the SOM with the COMPosite indicator. GENERATE OPTIONS=CSOM is normally invoked when a GENERATE CTLE is performed to generate the Composite Rules Load Module, but it can be used at any time a GENERATE statement is permitted to manipulate those SOM settings. A prerequisite for this function is that the TRXID must have been added to the SOM, either in the current Rules Generator run or in a previous one.

The following example illustrates the situation where the SOM entries for the affected transactions were created/updated during a previous run:

```
SYSTEM SYSID=SAMP,PG=ZZ,....
   SEGMENT ....
      FIELD ....
   etc.
GENERATE TRXID=AA,OPTIONS=CTLE,...
GENERATE TRXID=AB,OPTIONS=CTLE,...
GENERATE TRXID=AD,OPTIONS=CVALL,...
GENERATE TRXID=AC,OPTIONS=CTLE,...
GENERATE OPTIONS=SOM,...
GENERATE OPTIONS=CSOM,COMP=YES,CTRX=(AA,AB,AC,BB,BA,BC)
```

Transactions AA, AB, AC are generated with COMP=YES. In addition BB, BA, BC have composites specified, and their indicators are set on in the SOM by the Rules Generator, though no other reference to them is made in this run. Observe that AD was not selected for composite, though its SOM entry was updated (the only reason for the GENERATE SOM statement).

The following example illustrates the order of entries in a complex situation: where the transaction is being generated and is having a composite created during the current Rules Generator run:

```
SYSTEM SYSID=SAMP,PG=ZZ,....
   SEGMENT ....
      FIELD ....
   etc.
GENERATE TRXID=AA,..                  Generate the ITR normal TRXID
GENERATE OPTIONS=SOM,...              Update the SOM for TRXID=AA
GENERATE TRXID=AA,OPTIONS=CTLE,..     Now generate the composite
GENERATE OPTIONS=CSOM,COMP=YES,       Update COMP indicators in
         CTRX=(AA,AB,AC,BB,BA,BC)       ...those SOM entries
```

Transaction AA is generated and its SOM entry is updated (no composite). Now AA is generated as a composite (CTLE), the output of which is only the composite rule load module. To update the SOM entry (which was created by the previous GENERATE), the GENERATE CSOM entry is required. In addition to AA, the rest of the transactions named in the CTRX operand have the Composite indicators set in their SOM entries.

The Rules Generator processes the SOM in this way. When a transaction is added to the SOM, all flags are initialized and subsequent processing of the SOM does not disturb any flags set unless specifically requested. That is, the flags set by CSOM are initialized when the transaction is added to the SOM, but are only manipulated by GENERATE OPTIONS=CSOM.

Note: The setting of the composite indicator in the SOM does not guarantee that a composite load module exists. It results in the generation of the appropriate conversational composite load module name and the BLDL for it at execution time. If composite processing is not completed (by a GENERATE CTLE option), the BLDL fails and normal rules processing occurs.

Multiple GENERATE CSOM statements may be specified. In order to demonstrate the flexibility of the CSOM function, the following example is used:

```
GENERATE OPTIONS=CSOM,COMP=YES,CTRX=(ALL)
GENERATE OPTIONS=CSOM,COMP=NO,CTRX=(BB,CC,DD,EE)
```

The user may wish to do this if he has a large system with many transactions, most of which he wishes to set one way. He could use the ALL parameter on the first statement and specify the ones he does not wish to handle in that manner on the second statement. This example is one which could be used when a system is being moved from test to production. Another use lies in the conversion of composite rules generated in releases prior to IMSADF II Version 2 Release 1.

LINK-EDIT GENERATION OPTIONS

Standard Processing Transaction Drivers

The Rules Generator will link-edit executable transaction drivers which will operate with a common set of modules to process a transaction. A Standard Processing driver, under Conversational or Nonconversational processing, may process any number of IMSADF II transactions under one IMS/VS TRANSACT/APPLCTN definition. Each transaction driver is associated with a similarly named PSB, which contains up to 120 data base PCBs. Each transaction driver is related to an IMS/VS cluster code (SOMTX operand) which becomes part of the link-edited module name. Standard Processing transaction drivers are link-edited by the following OPTION parameters:

Note: A maximum of 300 transactions are allowed for each SOM rule for a given system ID.

```
OPTIONS= {STLE}                    (Conversational)
         {NCLE}                    (Nonconversational)
```

The GENERATE statement also specifies PGMID=tx, where tx is the cluster code desired. For example,

```
GENERATE    OPTION=STLE,PGMID=tx,SHTABLE=(IV),MAPTABLE=(M1)

GENERATE    OPTION=NCLE,PGMID=tx,SHTABLE=(IV),MAPTABLE=(M1)
```

Additional operands may be required to complete the link-edit process. These options, which are explained later in this chapter, are:

SHTABLE - Specifies all data base segment IDs used by the Standard Processing Routines.

MAPTABLE - Specifies all mapping segment IDs used by the Standard Processing Routine.

Special Processing Transaction Drivers

A Special Processing transaction driver, under Conversational or Nonconversational processing, will process a single transaction. These drivers are link-edited with a Special Processing Routine to form a unique executable module. Each Special Processing transaction driver is named by the PGMID=xx operand, where xx is the Special Processing transaction (TRXID). The COBOL, PL/I or ASSEMBLER special processing routine should be compiled and link-edited into a load library with the name ssssUxx, where ssss is the SYSID and xx is the TRXID. The Rules Generator includes (INCLUDE) the SPR from SYSLIB.

Special Processing transaction drivers are link-edited by the following OPTION parameters.

```
OPTIONS= {SPLE}                    (Conversational)
         {TPLE}                    (Nonconversational)
```

For example,

```
GENERATE OPTIONS=SPLE,PGMID=xx,LANG=COBOL,SHTABLE=(IV),
         MAPTABLE=(M1)
```

Additional operands may be required to complete the link-edit process. These options, which are explained later in this chapter, are:

SHTABLE - Specifies all data base segment IDs used by the Special Processing Routines.

MAPTABLE - Specifies all mapping segment IDs used by the Special Processing Routine.

```
OPTIONS=BDLE                       (Batch)
```

In addition, Batch transaction drivers can process both Standard and Special Processing transactions through a common driver module. The OPTION parameter BDLE is used to specify a batch transaction driver link-edit. PGMID is used to name the module. For example:

```
GENERATE OPTIONS=BDLE,PGMID=xx,LANG=COBOL,SHTABLE=(IV),
         MAPTABLE=(M1)
```

Additional operands may be required to complete the link-edit process. These options, which are explained later in this chapter, are:

SHTABLE - Specifies all data base segment IDs used by the batch transaction driver and any Special Processing Routines that issue SEGHNDLR calls.

MAPTABLE - Specifies all mapping segment rules used by all Special Processing Routines that issue MAPPER calls.

SLRTABLE - Specifies all pseudo segment IDs used by the batch transaction driver.

ITTABLE - Specifies all Input Transaction Rules used by the batch transaction driver.

SPTABLE - Specifies all transaction IDs for Special Processing transactions used by the batch transaction driver.

SPNAMES - Specifies all Special Processing routine names used by the batch transaction driver. This operand is required, in conjunction with the SPTABLE operand, if the names of the Special Processing routines contained in an object or load library do not follow the standard IMSADF II Special Processing naming conventions. The IMSADF II Special Processing naming convention is:

```
ssss - Application system ID

   u - Special processing module name

  xx - Special processing transaction ID
```

When SPNAMES is specified with SPTABLE a one to one positional relationship must exist between their parameters.

```
EXAMPLE:
  SPNAMES = (MFC1UMM,ADF3UZZ,SPR0001)
  SPTABLE = (MM,ZZ,TT)
```

The above example causes the three Special Processing routines to be included in the link-edit and associates them with their corresponding batch Input Transaction Rule as follows:

SPNAME(special processing routine)	SPTABLE(Batch ITR)
MFC1UMM	ssssBMM
ADF3UZZ	ssssBZZ
SPR0001	ssssBTT

If SPNAMES was not specified in the above example the Special Processing routine names included would be:

special processing routine	Batch ITR
ssssUMM	ssssBMM
ssssUZZ	ssssBZZ
ssssUTT	ssssBTT

SPTABLE - Specifies all transaction IDs for Special Processing transactions used by the batch transaction driver.

COMPOSITE RULES LOAD MODULE

```
OPTIONS= {CTLE } (Conversational)
         {NTLE } (Nonconversational)
```

A composite rules load module may be created which contains all rules required to process an IMSADF II transaction. The following rules may be included in a single composite rules load module:

Rule Type	How Specified
Input Transaction Rule	ITTABLE=XX
Segment Layout Rule(s)	SLRTABLE=(SEGID,SEGID,...)
Segment Handler Rule(s)	SHTABLE=(SEGID,SEGID,...)
Mapping Rule(s)	MAPTABLE=(SEGID,SEGID,...)
Output Format Rule(s)	OFRTABLE=(SEGID,SEGID,...)
Audit Static Rule(s)	ASRTABLE=(AGROUP,AGROUP,...)

See "Preload Rule" on page 2-99 for an explanation of these operands.

A conversational composite rules load module is generated as follows:

```
SYSTEM          SYSID=ABCD
GENERATE        OPTIONS=CTLE,TRXID=CM,PGROUP=AA,
                IT=CM,
                SL=(P1,A2,A3,A4),
                SHT=(A2,A3,A4),
                MAP=M1,
                ASR=(YYYY,ZZZZ)
```

A composite rules load module with the name (ABAACM##) is created and contains the following rules:

```
ITT - ABCDAACM
SLR - ABCDSRP1,ABCDSRA2,ABCDSRA3,ABCDSRA4
SHT - ABCDSA2,ABCDSA3,ABCDSA4
MAP - ABCDURM1
ASR - ABCDYYYY,ABCDZZZZ
```

A nonconversational composite rules load module is generated as follows:

```
SYSTEM          SYSID=ABCD
GENERATE        OPTIONS=NTLE,TRXID=CM
                IT=CM,
                SL=(P1,A2,A3,A4),    * P1 is a Pseudo Segment
                SHT=(A2,A3,A4),
                MAP=M1,
                ASR=(YYY,ZZZ)
```

The generated composite rules load module is named AB#TCM#

If a composite rules load module does not contain all the rules needed for the transaction, then normal loading occurs for those rules. If composite rules load modules are used the same rule may reside in multiple load modules. If a rule is changed, the user is responsible recreating all load modules which contain that rule. See the IMS Application Development Facility II Version 2 Release 2 Installation Guide for additional procedures for building a composite rules load module.

DATA DICTIONARY EXTRACT

OPTIONS=ADFX

The OPTIONS=ADFX operand is used to invoke the IMSADF II Extract Processor while the Rules Generator is active. The extract module obtains system, segment, field and generate information from the Rules Generator Tables and writes the information to two files. The first file contains only system, segment, field information for use by the ADFIN Data Dictionary Format Processor. It has a user specified ddname with a default of DDADFX, a DSORG of PS, and an LRECL of 256. The other file is used to create the ISPF Tables for the ISPF Interactive Front End. It has a ddname of ISPF, a DSORG of PS, and an LRECL of 1200. The format of the new option is:

```
GENERATE OPTIONS=ADFX,DDADFX=(ddname),SEGMENTS=(segid,segid,...),DECKS=NO
```

where:

DDADFX=

is the ddname for the data set used to store the IMSADF II Extract Data for the ADFIN Data Dictionary Format Processor. The ADFIN Processor will use this information to produce dictionary commands to update the system, segment, and field information in the Data Dictionary.

This is the name of the DD statement in the rules generator JCL to be used by the IMSADF II Extract processor for the Data Dictionary. The default ddname is DDADFX.

Note: This does NOT modify the ddname of the Extract data set used to create the ISPF tables.

SEGMENTS=

is a list of the segments to be updated in the Data Dictionary. These MUST be data base segments. If not specified, all data base segments will be updated in the Data Dictionary.

Note: ALL segments (not just data base segments) are ALWAYS extracted for the ISPF tables.

DECKS=

allows the user to set the DECKS operand for the remainder of the Rules Generator run. The default is NO.

Note: The OPTIONS=ADFX Generate statement must be specified after the last Field statement and before any other Generate statement in the Rules Generator source for this option to work correctly. The SYSTEM statement in the source being used must be hard-coded (i.e., it cannot be an INCLUDE statement).

For further information regarding the Extract processing, refer to IMS Application Development Facility II Version 2 Release 2 Data Dictionary Extension User's Guide.

GENERATE STATEMENT OPERANDS

Except where noted, the operands that follow apply to all three processing modes, Conversational, Nonconversational, and Batch.

OPERANDS (in alphabetical order):

ADDMODE= {**NO** | **YES**} **(Conversational only)**

The ADDMODE operand applies to the Input Transaction Rule for Conversational Processing (OPTIONS=INTR) and specifies to the transaction driver what to do on an Add mode transaction. The default is ADDMODE=NO, which indicates that after a segment is added, the transaction mode changes to update (mode 5). ADDMODE=Y specifies that after the segment is added, the transaction mode remains as an add (mode 2 or 4).

[AEXIT=xxxxxxxx]

The AEXIT operand names a user-written Audit exit which is to be included in a transaction driver link-edit. The object or load module named by xxxxxxxx must exist in the data set pointed to by OBJLIB or SYSLIB.

AGROUP= {**YYYY** | **XXXX**}

The AGROUP operand specifies the last four characters of the Audit Group name contained in the Input Transaction Rule. This name is used to identify specific Audit/logic routines connected with a transaction(s). The format of the Audit Group name is ssssxxxx where,

ssss = the major application system identification specified in SYSID

xxxx = any four characters to identify the Audit Group in the Audit Data Base

The default value is YYYY. Each AGROUP operand specified on a GENERATE statement overrides the value specified for AGROUP on the SYSTEM statement.

ALANG= { **COBOL** / **ASMINT** / **ASMSTD** / **PL/I** }

The ALANG operand specifies the language in which the Audit exit is written. In addition, it indicates how an Assembler Language Module is called by the transaction driver, either directly or through an interface routine. The ASMSTD value is used when the audit exit is called directly by the driver. The ASMINT value is used when the audit exit is called through the interface routine. ASMINT should normally be used if the audit exit is written in Assembler language. The default is COBOL.

[ALIAS=(sysid,...)]

The ALIAS= keyword on the GENERATE statement will allow Segment Layout and Segment Handler rules to be built that will be effective for multiple SYSID's. This keyword accepts a list of SYSID's to create Linkage Editor ALIAS statements. The following GENERATE statement creates the following Linkage Editor output:

```
GENERATE OPTIONS=SGALL,ALIAS=(JLD1,JLD2,..)


ENTRY          XXXXS1A

SETSSI         0000AD01

ALIAS          JLD1S1A

ALIAS          JLD2S1A

NAME           XXXXS1A(R)

where          XXXX is SYSID.
```

The ALIAS keyword on the OPTIONS=STLE or OPTIONS=SPLE GENERATE statement produces conversational mini-driver members with aliases to support user clustering of IMSADF II transactions. (See "Running an IMSADF II Transaction Under Multiple IMS/VS TRANCODEs" on page 7-9 for details.) The following statement produces the mini-driver member "SAMPTOR" with the alias "SA4CTOR":

```
GENERATE OPT=STLE,PGMID=OR,ALIAS=SA4C
```

ASMREQ= { **NO** / **YES** }

The ASMREQ operand is used to specify whether or not the Assembler source form of the requested rules is to be created. The ASMREQ=YES operand value will cause the Rules Generator to produce Assembler source statements. The Assembler will then be invoked dynamically to produce object modules. An Assembler listing is produced to verify that the Rules Generator has built the expected results.

The ASMREQ=NO operand value specifies that an Assembler listing is not required and therefore the Assembler will not be invoked. The Rules Generator will build object modules directly. ASMREQ=NO provides a significant performance improvement in the CPU time required to generate the rules.

BYPASS= { **NO** / **YES** } **(Batch only)**

For batch processing, the BYPASS=YES operand instructs the Batch Driver to make two calls to the Special Processing Routine. The first call is after the DBPATH segments have been retrieved and the second is after the Input Transaction Data Mapper has mapped the changed fields into the SPA workarea for Batch processing. The default is NO.

BYPASS= { **NO** / **YES** } **(Conversational only)**

For conversational processing, the BYPASS=YES operand is used to specify that the Special Processing routine is to be called prior to the display of the Segment Display screen. The default is NO.

CHKPT= { **NO** / **YES** } **(Batch only)**

The CHKPT operand may be specified for the generation of a Batch Transaction Driver link-edit (OPTIONS=BDLE). CHKPT=YES indicates that requests for IMS/VS checkpointing may be performed during each execution of the Batch Driver Rule, either by explicit request from the batch input stream or on frequency of input transactions. The default value is NO. See the FREQ operand for additional Rules Generator specifications.

CNT= { **1** / **n** } **(Batch only)**

The CNT operand indicates to the Batch Transaction Driver that the input medium is cards (80-byte records) and how many cards make up an input transaction message for this Input Transaction Rule. The maximum value for CNT is 25 and the maximum size transaction message that the Batch Transaction Driver can process is 2000 bytes.

Field positions, which are described in the FLDPOS operand of the FIELD statement, are specified as if the card input is contiguous. For example, the second card read would have a beginning FLDPOS of 81. The third card would have a beginning FLDPOS of 161. If a CNT of 2 or 3 has been specified in the Input Transaction Rule and a particular transaction message only needs one card to complete its data entry, the end of message indicator $$ can be specified after the last data needed in the transaction message. The Batch Driver will pad the remainder of the area with blanks and process the transaction message.

This operand is valid only when a Batch Input Transaction Rule is requested (OPTIONS=BAIT). The default value is one card.

COMMLEN= { **0** / **n** } **(Conversational only)**

The COMMLEN operand is used to specify the length of a user-maintained communication area in the SPA. This area can be used to pass data between Conversational transactions. The limit is the maximum value that can reside in the SPA along with other SPA resident rules.

This parameter may also be specified during installation and indicates an area which is to be available to all transaction, beginning at sign-on time.

The value you enter must be large enough to contain the sign-on area, plus the application area.

Any specification for COMMLEN readjusts the area to be the MAXIMUM of the current and new values, not the sum of them. This may become critical if you have transactions within an application close to your maximum SPA workarea.

Note: A project/group switch resets the COMMLEN to the value set at installation.

In addition, your mapping of the communication area must take into account the sign-on area, as shown in the following figure:

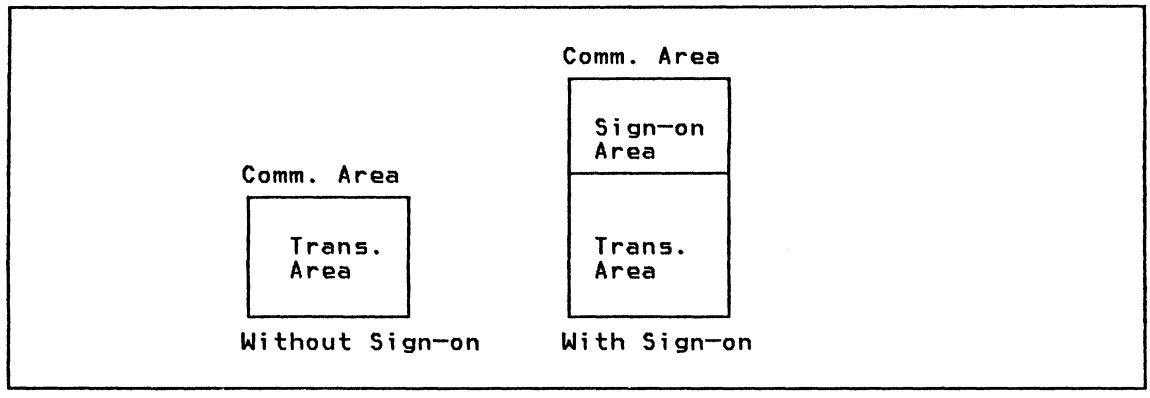


Figure 2-17. Communication Area User Mapping

[CURSOR=({fieldid / field.segid / adfname} ,...)]

The CURSOR operand specifies the field to which the cursor is initially positioned on each physical page. Parameter fieldid.segid specifies the field and segment ID where the cursor will start. The field must be defined as displayable. The parameter adfname specifies an IMSADF II field keyword where the cursor will start. See the definition of SIMAGE for the allowable keywords. This operand may be specified for either Conversational or Nonconversational segment display screens when the screen image option is specified. The default is the first unprotected data field. If there are no unprotected fields, the cursor will be positioned to the OPTION field for Conversational screens and the TRX field for Nonconversational screens. If more than one value is specified for CURSOR=, the relative position of each value in the list corresponds to the page number of the physical page on which the value (and therefore, cursor positioning) will appear. The maximum number of physical pages for one transaction is 22. Therefore, there can be a maximum of 22 cursor values.

DAMSG= {NO / YES}

The DAMSG operand is used to indicate that a deletion or addition notification message exists for the Input Transaction Rule requests on this GENERATE statement. The value YES will cause the executing transaction driver to search the Message Data Base for the applicable message when a deletion or addition of the target segment is made using this Input Transaction Rule. Note that the terminal operator entered transaction modes must be 1, 2, 3, or 4 for the notification message to be sent. The default value is NO, which indicates that no message needs to be sent upon deletion or addition of the target segment.

[DATACOMP=(segid,...)]

The DATACOMP operand specifies segments which are to be compared for change prior to update. This operand is valid only for conversational processing and should be specified for any segments which could have simultaneous updates by different users. If only one field in the segment is subject to modification, the DCFIELD operand on the SEGMENT statement can be used to verify only that field. Otherwise, the entire segment is compared. If the segment is a variable length segment, data compare of an individual field (DCFIELD) within the nonvariable portion of the VLS must be utilized.

If updates are immediately made to the data base prior to the data compare (HREP IMMED, ISRT IMMED, etc...) and the data compare fails, then the updates will be rolled out by an ABEND U0778.

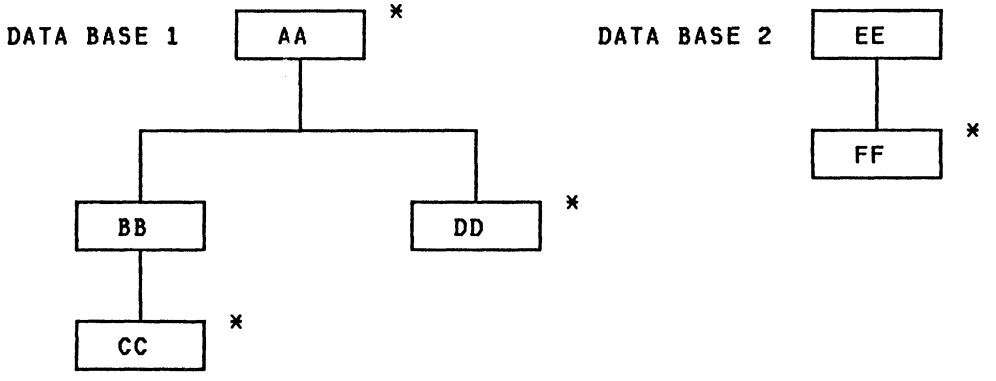
Data compare will be performed differently if null fields are present. The logic will be as follows:

1.....	compare all indicator variables in a single compare.	= goto 2	¬= error
2.....	is any indicator variable flagged as null?	yes goto 4	no goto 3
3.....	compare data areas.	= ok quit	¬= error
4.....	compare data areas field by field and indicator variable by variable	= ok quit	¬= error

[DBPATH=(segid,...)]

The DBPATH operand is used to specify the data base path(s) to be included in the Input Transaction Rule. Each segid specified is assumed to be the target segment (lowest level segment) of a hierarchical path in a data base. The Input Transaction Rule will contain an entry for each segment in each segment in each path which has a displayable field. These segments (with TYPE=DBS) will be loaded at the beginning of transaction processing. If a segment within a path does not have a displayable field, but must be loaded for Audit processing, then its segid should also be included.

The following example demonstrates the use of the DBPATH operand:



*=segment contains displayable fields

Figure 2-18. DATA BASE PATH Examples

```
DBPATH=(CC,DD,FF)
```

The above DBPATH specification causes segments AA, CC, DD, and FF to be loaded in the transaction. A Primary Key Selection screen can also be generated to allow entry of keys for segments (AA, BB, CC, DD, EE, and FF). If, in the above example, segment EE must also be loaded then the DBPATH operand would be:

```
DBPATH=(CC,DD,EE,FF)
```

Segments are loaded and updated by the transaction driver in the order in which they are specified in the DBPATH operand. In the above example, segments (AA,CC) are first loaded with a path call, then segment (DD) and finally segment (FF). If a segment's key is derived from another segment field (KEYBLD=), the source must be loaded first. In the above example, if the key of segment (DD) is derived from a field in segment (FF), then the DBPATH operand should be:

```
DBPATH=(CC,FF,DD).
```

The order in which segments are loaded is important if the Rules Generator is allowed to create the screen by the default options (DISPLAY=,MODE=). Fields will be displayed on the screen in the order in which the segments containing them are loaded. Also, fields are always audited in the sequence in which they are loaded. Fields are audited in the following order:

```
Pseudo segments (specified by TSEG)
     .
     .
     .
Data base segments (path 1) (e.g., AA and CC)
     .
     .
     .
Data base segments (path n) (e.g., DD and FF)
     .
     .
     .
Data base segments (specified by TSEG)
```

The screen defaults for each target segment in a path (lowest level segment) are DISPLAY=Y and MODE=5. All other segments default to DISPLAY=N (and therefore, MODE=6). This default is overridden by either DISPLAY/MODE specifications or explicit screen image definitions. The maximum combined number of segments that can be specified in the DBPATH and TSEGS operands is 100.

The following examples of transactions demonstrate various GENERATE configurations to create rules and screens for DATA BASE 1 and DATA BASE 2 pictured in Figure 2-18. Assume that SEGMENT, FIELD, and SYSTEM definitions for all six segments precede these GENERATE statements and that all Segment Layout and Segment Handler rules have been generated.

1. Standard Processing transaction to process path CC. Segments AA and CC have displayable fields.

   ```
   GENERATE OPTIONS=(INTR,KEYD,SEGD),TRXID=CC,DBPATH=CC
   ```

   Note: A TRXID may be the same as a target segment ID.

2. Same as transaction 1, but also load segment BB.

   ```
   GENERATE OPTIONS=(INTR,KEYD,SEGD),TRXID=01,DBPATH=(BB,CC)
   ```

3. Same as transaction 1, but reserves space for Auditor to load segment DD.

   ```
   GENERATE OPTIONS=(INTR,KEYD,SEGD),TRXID=02,
            DBPATH=(CC),TSEG=DD
   ```

4. Standard Processing transaction to process all three paths.

   ```
   GENERATE
   OPTIONS=(INTR,KEYD,SEGD),TRXID=03,DBPATH=(CC,DD,FF)
   ```

5. Special Processing transaction to process paths CC and DD. Also load segment BB and reserve space for a pseudo segment (PS).

   ```
   GENERATE OPTIONS=(INTR,KEYD,SEGD),TRXID=04,DBPATH=(BB,CC,DD
            TSEG=PS,SPECIAL=Y,KEYSL=Y
   ```

Two transaction drivers must be link-edited to process the preceding five transactions. A Standard Processing transaction driver will process transactions 1-4, and a Special Processing transaction driver will process transaction 5. The link-edits are specified as follows:

- Standard Processing transaction driver

  ```
  GENERATE OPTIONS=STLE,PGMID=xx
  ```

Note: xx is the cluster code (SOMTX) associated with transaction IDs CC, 01, 02, and 03.

- Special Processing transaction driver

  GENERATE OPTION=SPLE,PGMID=04,LANG=COBOL,MAPTAB=(M1)

  Note: The program ID and transaction ID are the same. MAPTAB specifies a mapping rule which will be used to access the pseudo segment area from the Special Processing routine. Assume this mapping segment has already been created (OPTION=SEGL).

The ID of each Input Transaction Rule generated above should also be reflected in the Secondary Option Menu Rule. This is accomplished by including the statement (GENERATE OPTION=SOM,UPDATE=YES) in each Rules Generator execution. This causes the IDs of any ITR generated to be added to the SOMR.

[DELETE=(trxid,...)]

The DELETE operand specifies that the transaction IDs should be eliminated from the existing Secondary Option Menu Rule.

DEVCHRS=(nn,nn,...) (1-22)

This keyword enables the Rules Generator to build the correct FEAT= option on the MFS DEV statement for transactions whose Segment Display screens are generated for both color and non-color. The values represent sets of terminal characteristics which are named in the IMS/VS terminal macro FEATS= keyword. The following are the values supported by DEVCHRS:

0	(IGNORE) default)		
1-10	(feature code F1 - F10 if used)		
11	(NOCD,PFK,PEN)	12	(CARD,PFK,PEN)
13	(NOCD,NOPFK,PEN)	14	(CARD,NOPFK,PEN)
15	(NOCD,DEKYBD,PEN)	16	(CARD,DEKYBD,PEN)
17	(NOCD,PFK,NOPEN)	18	(CARD,PFK,NOPEN)
19	(NOCD,NOPFK,NOPEN)	20	(CARD,NOPFK,NOPEN)
21	(NOCD,DEKYBD,NOPEN)	22	(CARD,DEKYBD,NOPEN)

Specify each set of characteristics desired within parentheses separated by commas. For each set you code, you must also code its corresponding DEVNAME/DEVTYPE value.

The parameter is not applicable to other screens or to text utility Segment Display screens and is ignored if present.

If you generate the same transaction for multiple terminals and DEVCHRS is not zero for ALL of them, you must code the DEVCHRS values for EACH terminal as they are specified in your IMS/VS Stage 1 System Definition.

If you do NOT generate the same transaction id (TRXID) for both color and non-color terminals, you should code the value for IGNORE(0).

There is a one-to-one correspondence among DEVNAME, DEVTYPE, and DEVCHRS, (for Segment Display screens only), as illustrated by the following example:

```
DEVNAME=(A2,A4,A3,A3)
DEVTYPE=(2,4,7,3)
DEVCHRS=(0,17,3,11)
```

Notes:

1. If you want screens generated for the installation terminal defaults only, do not code DEVNAME/DEVTYPE/DEVCHRS values. Any value(s) named on Rules Generator input will override all installation defaults.

2. This keyword is designed to replace DEVFEAT. Applications with DEVFEAT= will still work correctly, however, installation defaults will not be in effect. If DEVCHRS= and DEVFEAT= are specified for a transaction, DEVCHRS is used and DEVFEAT is ignored. If a user wishes to create screens for a subset of the default terminals, he must code all of the desired DEVNAME/DEVTYPE/DEVCHRS keywords (he no longer codes DEVFEAT).

```
                  {IGNORE                                                  }
DEVFEAT=          {1,2...10                                                }
                  {'[{CARD}][{,PFK  }][{,PEN  }]'                          }
                  {  [{NOCD}]{NOPFK }[{NOPEN }]                            }
                                {DEKYBD}
```

The DEVFEAT keyword enables the Rules Generator to build the correct FEAT= option on the MFS DEV statement for transactions that use both color and and non-color. IF DEVTYPE is specified with a value of 6 or 7, combined with another value, then a DEVFEAT keyword must be coded for each DEVTYPE value unless the monochrome terminals have been defined with the FEAT=IGNORE option on the IMS/VS system definition TERMINAL macro. The Rules Generator will be able to **build** correct default FEAT= options for for transactions that do not use a mixture of color and monochrome.

Examples

```
GENERATE    TRXID=PA,DBPATH=PA,
            DEVNAME=A2,DEVTYPE=2

GENERATE    TRXID=PA,DBPATH=PA,
            DEVNAME=A09,DEVTYPE=6

GENERATE    TRXID=PA,DBPATH=PA,
            DEVNAME=(2,A7,A5),DEVTYPE=(2,7,5),
            DEVFEAT='CARD,PFK,NOPEN',
            DEVFEAT=1,
            DEVFEAT='NOCD,PFK,PEN'
```

Only the third example is mixed with the second DEVFEAT required for color and corresponding to DEVNAME=A7 and DEVTYPE=7. The FEAT= values are described in the IMS/VS Message Format Services manual. Features coded must be consistent with those specified on the TERMINAL macro in the IMS/VS system definition. The Rules Generator does not check for consistency.

If DEVFEAT is not coded, the default value for a color terminal (DEVTYPE=6 or7) is taken from the DEVFEAT value on the DEFADF macro (see the IMS Application Development Facility II Version 2 Release 2 Installation Guide). The default value for a monochrome terminal is IGNORE. The Rules Generator does not place the IGNORE option on the MSG statement in the MFS MOD unless every DEVFEAT value on this transaction is set to IGNORE. If a format is built with a mixture of features, some of them FEAT=IGNORE, an IMS/VS error message is created when an attempt is made to use it online, unless FEAT=IGNORE is also specified on the IMS/VS system definition TERMINAL macro.

```
DEVNAME=([n  [,An][,...])   (2-9)
         An
```

The DEVNAME operand is used to specify the IMS/VS identification characters for terminal types that can be used with the transaction request on this GENERATE statement. The Rules Generator will generate a MFS device statement for each operand value. Screen formats (DFLD statements) will be duplicated for each device (see DEVTYPE for screen size considerations). The DEVNAME operand value

may be a two or three character identification starting with A or a 2 (for the 3277 display device model 2). The identification characters must match the IMS/VS SYSGEN selection for the appropriate device types.

There is a one-to-one correspondence among DEVNAME, DEVTYPE, and DEVCHRS (for Segment Display screens only), as illustrated by the following example:

```
DEVNAME=(A2,A4,A3,A3)
DEVTYPE=(2,4,7,3)
DEVCHRS=(0,17,3,11)
```

Note: If you want screens generated for the installation terminal defaults only, do not code DEVNAME/DEVTYPE/DEVCHRS values. Any value(s) named on Rules Generator input will override all installation defaults.

DEVTYPE=([n,...]) (2-9)

The DEVTYPE operand is used to specify the type of terminals that can be used with the transaction request on this GENERATE statement. Display device types 3275, 3277, and SLU type 2 are known to IMS/VS as the generic Model 3270, and as such, the Rules Generator does not distinguish between these types. The DEVTYPE operand values are used to set screen size characteristics and generally correspond to model numbers for the various display types. Operand values of 2, 3 and 4 represent models with screen widths of 80 characters and lengths of 24, 32, and 43 lines respectively. The operand value DEVTYPE=5 specifies a display model having a screen which is 132 characters wide and 27 lines in length.

The operand value DEVTYPE=6 specifies a 3279 color terminal Model 2B having a screen which is 80 characters wide and 24 lines in length. The operand value DEVTYPE=7 specifies a 3279 color terminal Model 3B having a screen which is 80 characters wide and 32 lines in length. The operand value DEVTYPE=8 specifies a 3290 Information Display Panel having a screen which is 160 characters wide and 62 lines in length.

The operand value of DEVTYPE=9 specifies a 5555 multi-station display having a screen that is 80 characters wide and 24 lines in length.

There is a one-to-one correspondence among DEVNAME, DEVTYPE, and DEVCHRS (Segment Display screens only), as illustrated by the following example:

```
DEVNAME=(A2,A4,A3,A3)
DEVTYPE=(2,4,7,3)
DEVCHRS=(0,17,3,11)
```

Note: If you want screens generated for the installation terminal defaults only, do not code DEVNAME/DEVTYPE/DEVCHRS values. Any value(s) named on Rules Generator input will override all installation defaults.

The smallest screen size of those specified for a transaction will determine the maximum screen length generated or page size allowed if screen image techniques are used. For the larger screen sizes, a duplicate of the DFLD statements for the smallest screen will be used. Separate transaction IDs may be specified for each device type (multiple GENERATE statements) if maximum use of each screen size is required.

The Sign-on screen generation provides spacing based on the screen width. The Secondary Key Selection screen only makes use if the first 80 characters of the screen width. The automatic generation of segment display screens provides one column, two columns, or compact data formats with spacing based on screen width. Screen image input for the 132- or 160-character screens can be entered following the GENERATE statement or can be included as a member of IMAGELIB. When entered following the GENERATE statement, two card

images are required for each screen layout line. The first 66 characters of each card are processed as image characters for the 132-character screen. The first 80 characters of each card are processed as image characters for the 160-character screen. Comments and the transaction field table statements are entered on one card.

The IMAGELIB data definition may be set up to describe an image library containing logical records of 132 characters in length. An editor with scrolling can be used to input 132-character line images directly into the IMAGELIB data set. In this case, comments may also be 132 characters long. If desired, screen images of 80-character screens may also be placed in the same library. Processing on DEVTYPE=2,3 or 4 would only include the first 80 characters of each record.

[DEXIT=xxxxxxxx]

The DEXIT operand names a user-written DL/I exit routine which is to be included in a transaction driver link-edit. The object or load module named by xxxxxxxx must exist in the data set pointed to by OBJLIB or SYSLIB.

DISPNAME= {MO | PG} **(Conversational only)**

The DISPNAME operand is used to specify the naming convention for the Segment Display Screen. If DISPNAME=PG is specified, the Project/Group characters are placed in the third and fourth positions of the screen name. Each Conversational Input Transaction Rule can then have a unique I/O screen as follows:

Input Transaction Rule name	sssspgxx
Segment Display Screen MOD name	sspgxx01

The default value of DISPNAME is used to specify a Segment Display Screen MOD name with the following format:

ssMOSxx1

This is the format for the installed IMSADF II screen names since only a single Project/Group need access each target segment in the IMSADF II dynamic rules data bases. Note that in changing from one naming convention to another for any target segment, the old screen members in MFS libraries are not automatically replaced. In those case, the Message Format Service facility should be run to scratch undesired indexes and members.

If DISPNAME=PG is specified then the label on the MFS FMT statement has the following format: sspgtt

ss - first two characters of the major application system ID
pg - two-character project/group code
tt - two-character transaction ID

This format does not allow for the use of MFSTRLR. Therefore, using MFSTRLR and specifying DISPNAME=PG could result in multiple MID/MOD members but only single DIF/DOF members in the MFS format library.

DKEY= {YES | NO} **(Conversational only)**

The DKEY operand allows the user to control the display of the KEY on the Segment Display Screen. By specifying DKEY=NO, the user can inhibit the key from being displayed on the Segment Display Screen. If DKEY=YES or if no DKEY keyword is specified on the Generate Statement, the Key will be displayed.

DLANG= { **COBOL** / **ASMINT** / **ASMSTD** / **PL/I** }

The DLANG operand specifies the language in which the DL/I exit is written. In addition, it indicates how an Assembler Language Module is CALLED by the transaction driver, either directly or through an interface routine. The ASMSTD value is used when the DL/I exit is CALLED directly by the driver. The ASMINT value is used when the DL/I exit is CALLED through the interface routine. ASMINT should normally be used if the DL/I exit is written in Assembler Language. The default is COBOL.

[DLET=(segid,...)]

The DLET operand specifies segments described in the Input Transaction Rule which are eligible for deletion from either the Auditor or a Special Processing routine using the SEGUPDTE call. If a segment is not eligible for deletion, and the Auditor or Special Processing routine attempts to do so, an error condition will be returned. For a single path transaction, the target segment's default is delete eligibility. For a multiple path transaction, the DLET operand must be specified if any segments have delete eligibility. In addition, audit rules must be placed in the Audit Data Base to issue the delete call or the Special Processing routine must utilize the SEGHNDLR or SETFLAG call.

DLIEXIT= { **NO** / **YES** }

Invocation of a DL/I exit is controlled at the transaction level. Specifying DLIEXIT=YES for a transaction causes the Common Segment Handler module to invoke the DL/I exit that has been link-edited with the mini-driver. This operand can be used on those transactions that have OPTIONS= INTR, CVALL, TPIT, TPALL, or BAIT. See the topic "DL/I Exit Examples" on page 6-44, for examples of how to use DL/I exits.

DTRAN= { **YES** / **NO** } **(Conversational only)**

The DTRAN operand allows the user to control the display of transaction code on the Segment Display Screen. By specifying DTRAN=NO, the user can inhibit the current transaction mode and ID from being displayed on the Segment Display Screen. If DTRAN=YES, or if no DTRAN Keyword is specified on the Generate Statement, the transaction code will be displayed.

DTWINC= { **YES** / **NO** } **(Conversational only)**

The DTWINC operand flags the Auditor whether to perform a validity check on the keys of twin segments that are to be inserted. The default is DTWINC=YES.

DYNLOAD= { **NO** / **YES** } **(Conversational Special Processing only)**

Special processing IMSADF II transactions can be started from the same IMS/VS transaction in the same manner as standard processing transactions. However, since the special processing transaction involves a user special processing routine, it must be packaged to be accessible. This packaging is done by the Rules Generator according to parameters on GENERATE statements with OPTIONS SPLE, STLE, and CVALL. The DYNLOAD and PGMID parameters on the SPLE and STLE link-edits and the DYNLOAD, SOMTX= parameters on a Conversation Input Transaction Rule Generate (CVALL) control the packaging.

DYNLOAD=YES indicates that the SPR is to be dynamically loaded and can participate in clustering. The load module created from the SPLE GENERATE is ssssZxx where ssss is the IMSADF II transaction ID (TRXID) and xx is the special processing program ID (PGMID). The

special processing module is clustered with the transaction driver created from an STLE GENERATE with module name ssssTtx where ssss is the IMSADF II transaction ID (TRXID) and tx is the cluster code. The cluster code matches the SOMTX of the Conversational Input Transaction Rule Generate (CVALL). DYNLOAD=YES on CVALL GENERATE indicates in the Secondary Option Menu Rule that this special Processing TRXID is to be scheduled under the Standard Processing IMS/VS transaction (ssssTtx), where tx is the SOMTX. Prerequisite for clustering of SPR modules is an STLE link-edit GENERATE statement with DYNLOAD=YES. See "Conversational Transaction Clustering" in Chapter 7, "IMSADF II under IMS/VS" for an example.

EOF= {NO | YES} **(Batch only)**

The EOF (end of file) operand may be specified when a Batch Input Transaction Rule is requested (OPTIONS=BAIT). The default is NO.

When OPTIONS=BAIT, a value of YES in this operand causes the Batch Transaction Driver to make a final call to the appropriate Special Processing Routine prior to termination to close OS/VS data sets. SPAFIRST is set to -1 to indicate that this is a final call.

FREQ= {0 | n} **(Batch only)**

The FREQ operand may be specified in conjunction with CHKPT=YES for the generation of a Batch Transaction Driver link-edit (OPTIONS=BDLE). It indicates the frequency of valid (syntactically correct) input transactions between checkpoints. If FREQ=0, checkpoints will be taken on an explicit request from the input stream. FREQ=n will be ignored if CHKPT=NO. The default value is 0.

[IMAGE=xxxxxxxx]

The IMAGE operand is used to include previously defined screens in an IMAGELIB. A GENERATE statement with an IMAGE operand must not be used as a part of a member of an INCLUDE statement because nested INCLUDES are not supported.

[ISRT=(segid,...)]

The ISRT operand specifies segment ISRT eligibility under the following conditions:

1. When the transaction mode is DELETE or UPDATE and this segment is not found, the segment will be ISRTed into the data base if any data has been entered by the user and the user has ADD authority for this TRXID. If this is a Conversational application, secondary key selection will be displayed for this segment only if no key was entered or a greater than (>) was entered.

2. When the transaction mode is ADD and this segment is not found, this segment and its dependents are eligible for a path ISRT. If this is a Conversational application, secondary key selection will be displayed only if a > is entered in the key field.

3. When the transaction mode is RETRIEVE and the segment is not found, the Data Display Screen is displayed with the segments in the path in which they were found.

The target segment of each path assumes ISRT eligibility if the transaction mode is ADD. ISRT eligibility specified for a parent segment will propagate down to all dependents. When the transaction mode is UPDATE and no segments are found for a complete path with ISRT eligibility, the mode is automatically changed from UPDATE to ADD.

[ITTABLE=(trxid,...)] **(Batch only)**

The ITTABLE operand is used to specify the set of non-Special

Processing Batch Input Transaction Rule transaction IDs that this Batch Driver load module generation may process. This operand creates a table containing all the Input Transaction Rule names to which this Batch Driver will have access. Only applicable for OPTIONS=BDLE.

KANAME= {**ALT** | **STD**}

The KANAME operand is used to specify whether Key Audits in the transaction are to be accessed under the Root Key 'SSSSYYYYSSXXFFFF' when KANAME=ALT is used. If the current naming convention of 'KEYAUDITSSXXFFFF' is desired then KANAME=STD maybe specified. The default value for KANAME is the value specified on the SYSTEM statement. If not entered on SYSTEM statement, the default will be set at installation time. Refer to "Audit/Logic Rules" on page 5-8 in Chapter 5, "Audit Logic Processing."

KEYSL= {**YES** | **NO**} **(Conversational only)**

The KEYSL operand is used to specify whether Key Selection is required for the transaction. If YES is specified, normal Key Selection will occur. If NO is specified, Key Selection will be bypassed and control will be passed directly from the Secondary Option Menu routine to the Transaction Driver. The default value is YES if DBPATH is specified; otherwise, the default value is NO.

LANGUAGE= {**COBOL** | **ASMINT** | **ASMSTD** | **PL/I**}

The LANGUAGE operand is used to specify the language in which the Special Processing routine is written. In addition, it indicates how an Assembler language module is called by the transaction driver, either directly or through an interface routine. The ASMSTD value is used when the user's module is called directly by the driver. The ASMINT value is used when the Special Processing Routine is an Assembler Language module called thru the interface routine. The COBOL value is used when the module is a COBOL program. The PL/I value is used when the module is a PL/I program. This parameter is used on the Input Transaction Rule and Linkage Editor GENERATE Statements when a special processing routine is included.

LINKLIB= {**SYSLMOD** | **XXXXXXXX**}

The LINKLIB operand specifies the name of a data set to receive the output of the rules generation or link-edit process. If no LINKLIB is specified, then the default data set is SYSLMOD. LINKREQ is automatically set to YES if the LINKLIB value changes from the previous LINKLIB value. The default (SYSLMOD) is not considered as a previous LINKLIB value.

LINKREQ= {**NO** | **YES**}

LINKREQ indicates that a link-edit is to be performed for all rules generated to this point. This operand is used if different libraries will receive the load module output. For example, rules may be stored in one library and Special Processing programs in another. The output library for the link-edit is SYSLMOD unless the LINKLIB operand is also used with the link-edit request, (see LINKLIB operand). LINKREQ is automatically set to YES when LINKLIB changes, when the generation request is a Secondary Option Menu rule or preload rule, or when all input has been read in by the Rules Generator. The link-edit performed at the end of input will use the LINKLIB value of the last GENERATE statement. If LINKLIB was not specified on the last GENERATE statement SYSLMOD will be used as the output library.

LRULE= { **NO** / **ALT** / **YES** }

The LRULE operand specifies how the audit rule data is retrieved. A YES value indicates that the audit descriptors are retrieved from a static audit load module which is loaded by Audit Group name at the start of execution. The value is generated as part of each Input Transaction Rule. Specification of the LRULE operand on the GENERATE statement overrides the system value for the current GENERATE statement. The default is NO, which means that the Audit Descriptors are to come from the dynamic Audit Data Base. A value of ALT causes the Static Rule to be retrieved. If the Static Rule cannot be found or specific audits are not present in the Static Rule, the Audit Data Base is accessed for the Dynamic Rule. LRULE=YES will yield maximum performance in the Auditor.

[MAPTABLE=(segid,...)]

The mapping rule table operand of the GENERATE statement is used with each OPTIONS=SPLE, TPLE or BDLE operand to provide mapping rule IDs for the Special Processing link-edit. Each mapping rule ID referred to in the Special Processing Routine may be specified. The Rules Generator will assemble and include a mapping rule table in the Special Processing load module.

For clustered Conversational Special Processing transactions, MAPTABLE can be specified on the STLE GENERATE and should include the mapping rule requirements for all clustered SPRs.

MAPTABLE is an optional operand. If the SPR issues a MAPPER call and the Mapping Rule is not found in MAPTABLE, the transaction driver will access the rule dynamically, via the Composite Rules load module, the Preload list, or a LOAD.

MAXKEY= { **50** / **n** }

The MAXKEY operand may be used to set the length of the screen area for entering key data directly in the KEY: field on the segment display screen. The default is 50 characters. For non-screen image input a MAXKEY value of 50 or less will result in the SYSMSG field being placed on line 5. A MAXKEY value greater than 50 will result in a SYSMSG field being placed on line 2 so that KEY: may use a contiguous field of up to 100 characters. For screen image input (SPOS=SIMAGE operand of GENERATE statement) the current MAXKEY length will be used to determine if any screen fields overlap. If &KEY is not specified in a screen image, a default of 50 will be used and the KEY: field will be placed on line 24. Specifying this operand overrides the value specified on the SYSTEM statement. The maximum key length is 100.

MODNAME=xxxxxxxx **(Nonconversational only)**

Any one- to eight-character name. The MODNAME value is the name specified by the end user to invoke the transaction. The default name is ssORttOa. (See Appendix C, "Naming Conventions.")

Do not use names which:

- are fewer than eight characters and end with the letter 'M'
- are eight characters with the letters 'BD' as the fifth and sixth characters.

MSGDB= { **0** / **n** }

The MSGDB operand is one to four numeric characters to specify the message number to be returned by IMSADF II when a DL/I call returns a non-blank status code other than "GE". The MSGDB operand is stored in the Segment Handler Rule and refers to a message number in the Message Data Base. Refer to the MSGNF operand for more detail on the message number. The default value is 0.

MSGNF= { **0** / **n** }

The MSGNF operand contains one to four numeric characters to specify the message number to be returned by IMSADF II when the requested data base segment is not found. The message number is stored in the Segment Handler Rule as a binary constant and refers to a message number in the Message Data Base. The message text is located in the Message Data Base through the application system identification and the message number. The application system identification and the message number make up the DBD sequence field of the root segment. The sequence field has the format:

```
ssssnnnn

where   ssss = the major application system identification
        nnnn = four digits right justified with leading zeros
```

The message number returned from IMSADF II will be right justified and will be preceded by leading zeros when the search key is built. The text is then located in the dependent System (SY) segments. The default value is 0.

Note: MSGDB and MSGNF are valid for special processing routines that issue SEGHNDLR calls. The IMSADF II driver will issue the specified message number if the proper conditions are met.

[OFRTABLE=(segid,...)]

The OFRTABLE operand specifies the Output Format rules which are loaded through the Preload or Composite Load Modules. Each SEGID entry specifies an Output Segment ID for which an Output Format Rule Exists. If the SEGID specifies exactly 2 characters, it is used as the Output Segment ID. If more than two characters are specified (eight-character maximum), it is used as the name of the Output Format Rule.

[ORID= { **segid** / **xx** } **]**

When used in conjunction with the generation of a nonconversational input transaction rule, the ORID operand is used to specify that the Nonconversational Driver is to generate an MFS output message to the entering terminal using the specified Output Format Rule. The segid specifies the two-character ID of the segment that was used to create the Output Format Rule. This operand is only applicable where a Nonconversational Input Transaction Rule generation is requested (OPTIONS=TPIT).

When used in conjunction with Options=OMFS, the two-character ID (xx) used on the ORID will be used by the Rules Generator in naming the Output Format Rule as well as the DOF and the MOD in the MFS source. It will also be used by the user in a later STX operand to identify which Output Format Rule to be used.

PFKDATA= { **NO** / **YES** } **(IMS/VS only)**

The PFKDATA=YES operand value is used to specify PFK literals as input transaction data. If PFKDATA=YES is specified and PFKLIT and PFKNUMB are not specified, the literals '01' thru '36' will be used. If PFKDATA=YES, a pseudo segment must be specified. The first field of this pseudo segment will receive the literal assigned to the PFK. The literal can be 1 to 22 bytes in length. Audits can be created to check the literal field and process accordingly.

The NO value indicates that the PFK literals contain only IMS/VS commands or paging requests. If PFKDATA=NO (or NO by default), then the default program function keys will be set to the following:

```
PFK 1,    PFKLIT=(NEXTLP)
PFK 2,    PFKLIT=('=-1 ')
```

```
PFK 3,    PFKLIT=('=1  ')
PFK 4,    PFKLIT=(ENDMPPI)
```

PFK 1, PFK 2, PFK 3, and PFK 4 are valid requests for paging in nonconversational processing. If multiple physical pages are required for conversational transactions, then the default PFK 4 is the only valid request for paging. Other physical paging requests for conversational transactions will be handled through the ACTION field. The default PFK paging requests can be overridden with the PFKLIT operand.

PFKLIT=('22 characters'),
..... **(IMS/VS only)**
PFKLIT=(10='19 char')

This operand may be used to specify literals that will be analyzed by IMS/VS or passed to IMSADF II when a program function key is entered instead of the device enter key. For Nonconversational processing these literals may be a combination of paging requests or user data to be interpreted by IMSADF II audit statements. For Conversational processing the literals can be either physical paging requests (only if multiple physical pages are in progress) or input data.

The format of the literal specification is in procedural order of the PFK numbers. In IMS 1.1.5 or later releases a PFK number may precede the literal so that PFKs may be skipped. A maximum of 22 characters may be specified including any key numbers. For PFKDATA=NO the default program function key literals for multiple physical pages are the following:

```
PFKLIT=(NEXTLP),    PFK 1
PFKLIT=('=-1 '),    PFK 2
PFKLIT=('=1  '),    PFK 3
PFKLIT=(ENDMPPI)    PFK 4
```

PFKNUMB=n **(IMS/VS only)**

The PFKNUMB operand may be used to specify the number of program function key literals to generate if PFKDATA=YES and PFKLIT is not specified. The default is 11.

PGMID=xx

The PGMID operand is used to specify the unique two-character identifier that is part of the naming convention for executable load modules. Conversational, Nonconversational, and Batch transaction driver executable load modules that are link-edited by the Rules Generator require a program identification. For Conversation or Nonconversational Standard processing, PGMID must match the cluster code (SOMTX) associated with the transaction IDs which use it. For Conversational or Nonconversational Special Processing load modules, the PGMID value must match the TRXID parameter used to generate the applicable Special Processing Input Transaction Rule. For the Batch Driver load module, the PGMID value can be any two characters that are unique from other Batch Driver program identification characters. Required for OPTIONS operand values of SPLE, TPLE, and BDLE.

[PGROUP=xx]

The PGROUP operand specifies the two-character Project/Group to be used in building the Input Transaction Rule name. The PGROUP is required only if OPTIONS=INTR,CVALL,TUALL, or TEXT on the GENERATE statement. If this operand is not specified on the SYSTEM statement, each GENERATE statement that requests an Input Transaction Rule must provide the specification for Project/Group. When this operand is is specified on a GENERATE statement, its value overrides the value specified for PGROUP on the SYSTEM statement.

[PHEADING='1 to 60 chars'] **(Batch only)**

The PHEADING operand may be specified for the generation of a Batch

Driver link-edit (OPTIONS=BDLE). The printer heading text will be included in the generation of a Batch Driver Rule as a header text for the transaction printer. A maximum of 60 characters may be entered. The default is a heading of one blank character.

POMENU= { ALL | ([A,B,C,D,F,H,I]) }

The POMENU operand indicates that a Primary Option Menu Rule is to be generated and specifies which options are to be displayed on the Primary Option Menu screen. This operand is ignored if it was already specified on a previous GENERATE or SYSTEM statement. A value of ALL indicates that all options A through I are to be displayed. Combinations of options can be specified. The options represent the following areas of processing:

A Project Message Sending
B Project Message Display
C Session Termination
D Transaction Selection
F Project/Group Switch
H User Message Sending
I User Message Display

Refer to the IMS Application Development Facility II Version 2 Release 2 User Reference manual for an example of the uses of the various options. The default value is ALL when OPTIONS=POM is specified.

[PRINTER= { 1 | 2 | 3 } **]**

The PRINTER operand may be used to direct output to a printer. This specification tells the Rules Generator which device characteristics to use for building the screen source. The values allowed and line widths used are 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

When the PRINTER= parameter is not used, default is the 3270 model 2 terminal. When a printer is specified, and the screen image follows the GENERATE statement, two card images are required for each printer layout line. The first 66 characters of the first card and the first xx characters of the second card are processed as image characters, where xx is 54 for the 3270 printers (printer=1 or 2) and xx=66 for the SCS1 printer (printer=3).

When a printer is specified, and the screen image is to be included from the IMAGE= parameter, then the screen image should represent the exact output layout. (i.e., the two card image is invalid.)

- Data cannot extend into column 120 for PRINTER=1 or 2 and column 132 for PRINTER=3.

  When IMSADF II builds the DFLDs for the screen image, whether the SIMAGE is in IMAGELIB or not, it uses a blank as a delimiter. If one line of the screen image is coded with literals to the allowable maximum length, then the following SIMAGE line may not begin in column 1. If so, IMSADF II will treat this card as a continuation and the DFLD will show an overlap. The same applies in reverse. If a line of the SIMAGE is to begin in column 1, then the previous line in the screen image may not be filled with literals in the last column. (i.e., the last column must be blank.)

[SEGMENTS=(segid,...)]

The SEGMENTS operand is used to specify one or more segment IDs to which the GENERATE statement applies. If this operand is not specified for the OPTIONS=SGALL value, all segments defined prior

to the GENERATE statement will be used as the default. This operand must be used with OPTIONS=SEGL or SEGH.

SFORMAT= { **DASH** | **LEFT** | **RIGHT** }

The SFORMAT operand defines the format of the literal fields to be generated for the Primary Key Selection and Segment Display screens. When screen image statements are provided for defining a transaction, the SFORMAT operand is not used for Segment Display screen generation (see SPOS=SIMAGE operand value of the GENERATE statement for screen image literal specifications). The Rules Generator builds an MFS literal and variable field for each FIELD statement requiring display on a Primary Key Selection or Segment Display screen. For Primary Key Selection, the display fields are the key fields specified with KEY=YES. For Segment Display, the display fields are those fields specified with DISPLAY=YES. Each literal is built from the value of SNAME as it is specified in the appropriate field definition. Screen centering is calculated based on the length of both the longest SNAME literal and the longest displayed field.

SFORMAT defines how the literal fields will appear on the screen. The value DASH specifies that dashes are to be added to make each literal length equal to the longest SNAME length plus one. The value LEFT indicates that the literal is left justified and followed by blanks. The value RIGHT indicates that the literal is right justified and preceded by blanks. The default value is DASH. This operand specified on a GENERATE statement overrides the value specified on the SYSTEM statement.

An example of the DASH operand value is as follows:

```
LITERAL 1------- data
LIT 2----------- data
LONGEST LITERAL- data
```

An example of the LEFT operand value is as follows:

```
LITERAL 1       data
LIT 2           data
LONGEST LITERAL data
```

An example of the RIGHT operand value is as follows:

```
      LITERAL 1 data
          LIT 2 data
LONGEST LITERAL data
```

Left and right justification applies to the literal area of an aligned screen when the automatic screen layout option is used for the Segment Display screen or for the Primary Key Selection screen. The length of the literal area of an aligned screen is the length of the longest literal to be displayed. When explicit field positioning is specified for the Segment Display Screen (SPOS=ROWCOL), the SFORMAT options apply to an area length between the two columns specified. area length between the two columns specified. If only one column is specified, the Rules Generator will use right justification for that field without regard to the SFORMAT option. If two rows are specified, one for the literal and one for the data area, the Rules Generator will use left justification.

[SHEADING='1 to 54 chars']

The SHEADING operand contains the text to be displayed on the first line of the Sign-on screen, Primary Key Selection screen, Segment Display screen, and Text Utility Screen. SHEADING text is enclosed in single quotation marks (' ') and can be from 1 to 54 characters in length. The text becomes a DFLD literal in the MFS source statements generated for the requested screens. The default value

is one blank. This operand specified on a GENERATE statement overrides the value specified on the SYSTEM statement.

[SHTABLE=(segid,...)]

The segment handler table operand of the GENERATE statement is used with each OPTIONS=SPLE or TPLE operand to provide Segment Handler Rule IDs for the Special Processing link-edit. Each segment handler ID referred to in the Special Processing Routine may be specified. This includes segments that are not in the key selection path included in the Input Transaction Rule and segments in the path for which replacements are made or twin processing is accomplished. The Rules Generator will assemble and include a Segment Handler Rule table in the Special Processing load module. The operand is also used with OPTIONS=BDLE to specify all data base segments which the link-edited batch transaction driver can access.

For clustered Conversational Special Processing transactions, SHTABLE can be specified on the STLE GENERATE and should include the segment handler rule requirements for all clustered SPRs.

SHTABLE is an optional operand. If the SPR issues a SEGHNDLR or SEGUPDTE call and the Segment Handler Rule is not found in SHTABLE, the transaction driver will access the rule dynamically, via the Composite Rules Load module, the Preload list, or a LOAD.

SIGNON= {NO / YES} **(Batch only)**

The SIGNON operand may be specified for the generation of a Batch Transaction Driver link-edit (OPTIONS=BDLE). It indicates that user authorization is performed for each execution of the Batch Driver Rule. SIGNON=YES specifies that a SIGNON transaction will be included in the batch input stream. The default value is NO. See "Sign-on Processing in Batch Mode" on page 6-36 for further explanation of Batch Sign-on.

The Sign-on screen may not be requested on more than 1 statement (SYSTEM or GENERATE).

SIGNON= {YES / NO / RACF} **(Conversational only)**

The SIGNON operand specifies whether or not a Sign-on screen is to be generated. The value YES indicates that screen source is to be generated for the Sign-on screen. The four-character major application system identification specified in the SYSID operand is the Sign-on screen name. The default value is YES if OPTIONS=SIGN is specified. If Sign-on screens are desired for terminals other than those specified as installation defaults, the GENERATE statement must be used. In those cases, the DEVNAME and DEVTYPE operands of the GENERATE statement must also be specified. The value of RACF will cause the Sign-on screen to be generated without the USERID field and during Sign-on processing the RACF USERID is used as the Sign-on USERID. The following statement will build the Sign-on screen with a RACF interface:

```
GENERATE OPTIONS=CVSYS,SIGNON=RACF ---
```

The Sign-on screen may not be requested on more than one statement (SYSTEM or GENERATE).

If Sign-on screens are desired for terminals other than those specified as installation defaults, the GENERATE statement must be used.

[SLRTABLE=(segid,...)]

The SLRTABLE operand is used with the OPTIONS=BDLE operand to specify all Pseudo segments which can be used by the link-edited batch transaction driver. SLRTABLE IDs are combined with SHTABLE IDs to form a complete list of all Segment Layout rules used by the

batch transaction driver. In addition SHTABLE forms a complete list of Segment Handler rules used.

SOIMAGE= { **NO** | **YES** }

SOIMAGE=YES indicates that a Sign-on screen is to be built using the screen image which immediately follows this statement. With SOIMAGE=YES on either the SYSTEM or GENERATE statements, you may not also request a Sign-on screen in both places.

If SOIMAGE=NO and a Sign-on screen is requested, a default screen will be generated. The default is SOIMAGE=NO.

The fields allowed on the Sign-on screen image are:

1. Literals - Literal data is entered directly in the screen image statements at the required screen locations.

2. System fields - Fields of the form &adfname may be used where adfname is a system field as described below. (The CURSOR operand on the GENERATE statement may refer to these fields for cursor positioning.)

   USERID six-character userid from the Sign-on profile. This field is optional for SIGNON=RACF.

   PROJECT one-character project identifier from the Sign-on profile.

   GROUP one-character group identifier from the Sign-on profile.

   LOCKWORD eight-character password.

   OPTION one-character option as would normally be chosen from the primary option menu.

   TRAN transaction mode and secondary option value. Format will be mss, where m is the current mode number chosen for the transaction and ss is the secondary option value, which can be either a target segment or a Special Processing ID.

   KEY concatenated key of the hierarchical path being displayed. Maximum length for the KEYID field on the Sign-on screen is 50 bytes. Key length cannot be altered by MAXKEY.

   SYSMSG 70-character system message field. This field will be displayed in high intensity and protected.

   SYSID four-character System ID, may eliminate the need for a different Sign-on screen for each system.

   LTNAME eight-character terminal identifier.

   TIME eight-character format HH:MM:SS (current time).

   DATE1 six-character format YY.DDD (current date).

   DATE2 eight-character format MM/DD/YY (current date).

   DATE3 eight-character format DD/MM/YY (current date).

   DATE4 eight-character format YY/MM/DD (current date).

   The first eight system fields are required. If they are not specified, an attempt will be made to default them to the last three lines as follows:

   ```
   Last Line     : SYSMSG.
   Last Line - 1 : OPTION,TRAN,KEYID.
   Last Line - 2 : USERID,PROJECT,GROUP,LOCKWORD.
   ```

If defaulting any of these fields causes an overlap with any other field on the screen, a terminating error message will be generated.

The last seven system fields are optional. The last six system fields are not applicable if ENVIR=CICS is specified on either the SYSTEM or GENERATE statements.

3. Control fields - Fields of the form &control specify screen image control parameters, where control is described below. In each case the & must be entered in column 1 and must be immediately followed by the control character.

   &* - Specifies that this is a comment statement. Comments follow the asterisk.

   &=nn - Specifies the number of blank lines to insert on the screen before the next line is created.

Position 1 of line 1 is reserved and not available to the user. Positions 1 through 4 of the last line are reserved for the size field and not available to the user.

The Sign-on screen will have the following default display attributes for these system fields:

field	default display mode
USERID	5 (display/modifiable)
PROJECT	5 (display/modifiable)
GROUP	5 (display/modifiable)
LOCKWORD	7 (non-display/modifiable)

Mode 7 may be specified for USERID, PROJECT, and GROUP system fields which causes these fields to be built as Non-Display/Modifiable. Mode 5 may also be specified for LOCKWORD which causes this field to be built as Display/Modifiable.

EXAMPLE -

```
         GENERATE OPTIONS=SIGN,SOIMAGE=YES,
                  SHEADING='BANKING SYSTEM'
                &DATE2                                &TIME
                                                      &SYSID
                BANKING SYSTEM

      ENTER YOUR 6 CHARACTER USERID:-------------------- &7USERID
      ENTER YOUR 1 CHARACTER PROJECT NUMBER:------------ &PROJECT
      ENTER YOUR 1 CHARACTER GROUP NUMBER:-------------- &7GROUP
      ENTER YOUR 8 CHARACTER PASSWORD--IF APPLICABLE:--- &5LOCKWORD

          OPTIONS                            TRANSACTION MODES

A = PROJECT MESSAGE SENDING                     3 - REMOVE
B = PROJECT MESSAGE DISPLAY                     4 - ADD
C = SESSION TERMINATED                          5 - UPDATE
D = TRANSACTION SELECTION                       6 - RETRIEVE
F = PROJECT GROUP SWITCH                 FOR OPTION - IDENTIFIER IS
H = USER MESSAGE SENDING                       D    - TRANSACTION ID
I = USER MESSAGE DISPLAY                       F    - PROJECT GROUP
                                          A,B,C,H,I - (NOT USED)

      TRX IS THE CONCATENATION OF TRANSACTION MODE AND IDENTIFIER
      OPTION:       TRX:       KEY:
&ENDS
```

In the above example, the system fields USERID and GROUP will be defined as Non-Display/Modifiable. The LOCKWORD will be defined as Display/Modifiable.

SOMEND= { **YES** / **NO** }

The SOMEND operand is used to specify whether a Secondary Option Menu Rule source deck that is to be generated (DECK=YES) is to have the LAST=YES keyword added to the last entry generated in the run. The default is YES. The NO value is used when the deck is to be combined with other decks.

SOMTX=tx **(Conversational only)**

The SOMTX operand specifies the IMS/VS transaction cluster code associated with the transaction ID (TRXID) for the Input Transaction Rule. This cluster code will be included with the transaction ID entry in the Secondary Option Menu Rule. This operand will override the SOMTX operand on the SYSTEM statement for this generation only.

SPECIAL= { **NO** / **YES** }

The SPECIAL operand specifies that the Input Transaction Rule generated is for a Special Processing transaction. The default is NO.

[SPNAMES=(sprname,...)] **(Batch only)**

The SPNAMES operand specifies all Special Processing routine names used by the batch transaction driver. Refer to Batch Transaction Driver link-edits for additional definition (OPTIONS=BDLE).

SPOS= { **AUTO** / **ROWCOL** / **SIMAGE** } **(Conversational only)**

SPOS=SIMAGE **(Nonconversational)**

The SPOS operand of the GENERATE statement specifies the type of field positioning that is to be used for generating the Segment Display Screen requested by this GENERATE statement.

When the screen image layout operand value (SIMAGE) is specified, a set of screen image statements must immediately follow the GENERATE statement (or its last continuation card). If the IMAGE=member operand is also used, then the screen image statements are retrieved from the specified member of the IMAGELIB DD library instead of the following GENERATE statement. In this case, the default value for the SPOS operand is SIMAGE.

When the AUTO value is active, the Segment Display Screen layout will be determined by the number of fields selected for the segment display screen. It will contain field data areas selected for display and their associated field literals, which are input through the SNAME operand of the FIELD statement. Field inclusion is accomplished by the DISPLAY=YES operand of the FIELD statement or by the field being in the target segment without a DISPLAY=NO value.

For automatic screen generation, field data areas are aligned into a one or two column screen format. If the number of fields is less than or equal to the number of lines available on the device and the lengths of the SNAME literals and the field data lengths permit, each field is placed on a separate screen line. If this is not the case, a two-column format or a compact format will be generated. In the two-column format, two consecutive short fields are placed on one line and long fields are placed on the next line or lines. Multiple physical pages are generated automatically to continue this format for all fields specified for the transaction. If long fields are present and SFORMAT=DASH is not specified, a more compact screen layout is generated. In the compact format, field literals are placed two bytes after the previous field data length if the literal and data length will fit on the line. Otherwise, the field literal will be used to start the next line.

The ROWCOL value is entered to request explicit screen positioning for a specific Segment Display Screen generation. The Rules Generator requires each field in the display to be specified with SROW and SCOL parameters. The default values for SROW and SCOL are zero, so that failure to provide an SROW or SCOL will result in a diagnostic message, 'SCREEN DATA CONTAINS INVALID ROWCOL', from the Rules Generator. The user can explicitly define the starting position in the segment display screen layout for each field literal and each field data area. Screen rows 6 through 24 are available for this purpose. The GENERATE SFORMAT options also are available when the literal and data areas are in the same row. If a single column is specified, right justification of the literal is assumed two screen positions preceding the column where the data area is to start.

Note that if segment data from the path to be displayed requires more than one screen for display, that data may have to be respecified with different row and column values in order to merge with other target field positions. This would apply if the user is not restricting screen layout rows with some convention. Note, too, that a segment may have row and column specifications for each display field and continue to use those fields in the automatic mode without the possibility of overlapping fields occurring. It is suggested that the user sketch a screen image layout for explicitly defined screens in order to avoid diagnostics due to overlapping fields on the screen.

An example of a portion of an explicitly defined screen is shown below.

FIELD	SFORMAT AVAILABLE
SROW=R,SCOL=C	RIGHT ONLY
SROW=(R1,R2),SCOL=(C1,C2)	LEFT ONLY
SROW=R,SCOL=(C1,C2)	LEFT OR DASH AS SPECIFIED

Example:

```
FIELD ID=A,L=4,SN='A NAME=',SR=6,SC=17

FIELD ID=B,L=2,SN='B NAME',SR=6,SC=(25,35)

FIELD ID=C,L=56,SN='C NAME',SR=(7,8),SC=(10,2)
```

will generate the following:

```
       Col.    Col.     Col.     Col.      Col.                 Col.
       2       10       17       25        35                   57
       |       |        |         |        |                    |
       |       |        |        _|        |                    |
       |       |        |       |          |                    |
       |       V        V       V          V                    |
Row 6  |       A NAME=  AAAA    B NAME---  BB                   |
Row 7  V       C NAME                                           V
Row 8  CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
```

The third method (SPOS=SIMAGE) for defining a Segment Display Screen consists of preparing a set of screen image statements that represent lines on a 3270 display device. This method may be used for Conversational screens and must be used for Nonconversational screens when OPTIONS=TPALL is specified. Also, this method must be used to create multiple-physical page segment display screens. See section "Screen Image Input" on page 2-101 following the end of the discussion about the GENERATE statement.

[SPTABLE=(trxid,...)] **(Batch only)**

The SPTABLE operand is used to specify the set of Special Processing transaction IDs that this batch transaction driver load module generation may process. The trxid value is the

two-character transaction ID for which a Special Processing Batch Input Transaction Rule exists. This operand creates a table containing all the Special Processing Routine names to which this batch transaction driver will have access. Only applicable for OPTIONS=BDLE and Preload Rule.

SQLCALL= { **(CSELECT,INSERT,CUPDATE,CDELETE,KSELECT1)** | **(SELECT,UPDATE,DELETE,KSELECT2,DSQLCALL)** | **(NONE)** } **(DB2 only)**

The SQLCALL operand specifies the SQL calls to be included in the Table Handler Rule.

- If SQLCALL is not specified then the 5 defaults CSELECT, INSERT, CUPDATE, CDELETE, KSELECT1 are taken.
- If SQLCALL is specified then all keywords including the defaults must be specified.
- The defaults may be entered individually or by specifying the DSQLCALL keyword (include all 5 defaults).
- If SQLCALL=NONE is specified then SQLUSER=YES must be specified and no other keywords are allowed.
- If SQLCALL=KSELECT2 is specified then the key field in the WHERE clause that will be manipulated by the SQL LIKE function must be the first key defined with an alphanumeric data type.

  User SQL calls may also be included in the Table Handler Rule for execution in the High Level Audit Language or in the SQLHNDLR Exit function.

SQLUSER= { **NO** | **YES** } **(DB2 only)**

A SQLUSER value of YES indicates user supplied WHERE clauses are to be included in this Table Handler Rule. Default value is NO. If YES, the user supplies a label, the SQL statement type, and a WHERE clause. These user augmented SQL statements may be executed during audits or exit SQLHNDLR calls. Labels cannot match any of the standard supplied SQL statements, named above. A maximum of 26 SQLUSER WHERE clauses can be specified for the Table Handler Rule. User SQL information is specified following the GENERATE OPTIONS=TABH statement, is terminated with &SQLENDS, and is input in the following format:

```
GENERATE OPTIONS=TABH,TABLES=XX,SQLUSER=YES

* LABEL  STATEMENT       CLAUSE
*        COLUMNS
* 1 to 8 10 to 17 19 to 71
*               16 CONTINUATION (16 TO 71)
* ------ -------- ------------------------------------------
XXSELECT SELECT   WHERE TABCOLUMN1 BETWEEN :ffff.xx AND :ffff.xx
                AND TABCOLUMN2 > :ffff.xx
XXUPDATE UPDATE   WHERE TABCOLUMN1 IN (:ffff.xx,:ffff.xx)
XXDELETE DELETE   WHERE TABCOLUMN1 <= :ffff.xx AND TABCOLUMN1 = 0
XXCURS1  CURSOR   WHERE TABCOLUMN1 LIKE :ffff.xx
XXCURS1O OPEN     XXCURS1
XXCURS1F FETCH    XXCURS1
XXCURS1U UPDATE   WHERE CURRENT OF XXCURS1
XXCURS1D DELETE   WHERE CURRENT OF XXCURS1
XXFURS1C CLOSE    XXCURS1
KSELECT SELECT    WHERE TABCOLUMN1 BETWEEN :ffff.xx AND :ffff.xx
&SQLENDS
```

Figure 2-19. SQLUSER Input Format

LABEL LABEL is a one- to eight-character name, where the first character must be alphabetic. This is the name that will be used in the High Level Audit Language SQL call or SQLHNDLR call.

Another SQLUSER option is to augment the Key Selection function. In this case, LABEL has the format KSELECTn, where n is 3 through 9. n=1 and 2 are reserved for IMSADF II standard key selection function. The Key Selection function is invoked if the terminal operator enters '>', '<', ' ', '_', an incorrect key, no key, or if primary key audit sets SPASQL. Remember the overall invocation control for secondary key selection is SKSEGS. Note that where key input is through the COFIELD vehicle with a character data type, the terminal input is retained in the COFIELD area of the SPA workarea.

STATEMENT SQL statements are:

SELECT

UPDATE

DELETE

CURSOR

OPEN

CLOSE

FETCH

CLAUSE CLAUSE will contain the WHERE clause for a SELECT, UPDATE, DELETE, or CURSOR statement and the CURSOR label in the OPEN AND CLOSE statements.

The WHERE statement may contain any of the DB2 comparison operators, predicate relational operators and calculated values. The host variables in the WHERE clause are represented through the IMSADF II column names (:ffff.xx). This name must be previously defined in this Rules Generator input stream. The ffff defines the one- to four-character IMSADF II column ID and must be followed immediately with .xx where xx is the two-character IMSADF II Table ID. The COLON is also required.

For the OPEN, FETCH, and CLOSE statements, the CLAUSE will contain the LABEL name of the applicable CURSOR.

The selected columns, SET clause, and VALUES clause are built by IMSADF II and are defined according to the IMSADF II Rules Generator Table definition, SQLUPD, SQLISRT, and SQLDIST operands.

For user key selection SQL specification (LABEL KSELECTn, SQL STATEMENT type SELECT), the WHERE clause may contain the KEY=YES host variables and pseudo segment host variables. If more than the keys are specified, the High Level Audit Language function to set SPAWHERE must be executed during a Primary Key Audit. SPAWHERE must reflect the host variables in the defined data format, length, and order.

User supplied key selection WHERE clauses must be able to reposition in the table for subsequent schedulings, when the terminal operator requests the next page of selections. If a row is selected for display, the row will be retrieved for display using the standard IMSADF II function CSELECT call. The keys used for this retrieval may be different from keys used for a user defined secondary key selection browse if non-unique keys were used for the secondary key selection browse.

User defined KSELECTn(n=3-9) functions only specify the SELECT function. Logic is included in the Table Handler Rule to DECLARE, OPEN, FETCH, and CLOSE CURSOR. These functions are manipulated by IMSADF II secondary key selection modules when KSELECTn is specified.

If additional function is required, native SQL calls should be issued in an audit or special processing exit and mapped back to the IMSADF II SPA workarea (Table ID or pseudo segment ID) via the MAPPER function.

Only those SQL calls that will be executed by the IMSADF II transaction drivers, should be included in the Table Handler Rule. The number of SQL calls determine the size of an application plan and the size of an application plan is an important consideration in the performance of the DB2 system.

[STX=({MFS|TRX},segid [,OK][,ER] [,{0|n}])]

The STX operand specifies that if the specified criteria are encountered, a message is to be generated and sent to a specified destination (transaction or terminal). The transaction type and output segment id are required positional parameters for each STX specification. The STX operand can be repeated on a GENERATE statement to account for different criteria combinations. In addition to the criteria specified as part of the STX operand an audit setting also is available to select a particular output format that will be generated as a secondary transaction. In that case, only the transaction type and output segment id are required as STX operand values. A maximum of 60 secondary transaction messages can be generated for each Input Transaction Rule.

For a description of the operand values, see "Secondary Transactions/Output Message Routing" on page 7-10 and "Secondary Transaction/Output Message Routing" on page 8-5.

TABLES=(id,...) **(DB2 only)**

TABLES specifies the two-character ID (or IDs) identifying the DB2 table/view required in generating Table Layout and Table Handler rules.

[TPITTABL=(trxid,...)]

The TPITTABL operand specifies the nonconversational Input Transaction Rules which are to be loaded through the Preload Rule. Each trxid entry specifies a two-character transaction ID for which an Input Transaction Rule exists.

TRXID=trxid

The TRXID operand specifies a unique IMSADF II transaction ID for the generated Input Transaction Rule. This ID is also included in the Secondary Option Menu Rule in Conversational processing.

TRXNAME='1 to 30 characters'

The TRXNAME operand specifies the transaction name which is displayed on the Secondary Option Menu selection, Primary and Secondary Key Selection screen and for the Segment Display screen if screen image is not specified. This name is included in the Secondary Option Menu Rule entry for the TRXID.

[TSEGS=(segid,...)]

The TSEGS operand is used during rule generation to specify that other data base segments and/or pseudo segments (including COMM segments are to be part of an Input Transaction Rule. This causes space to be reserved in the SPA segment area, or communications area and causes the Segment Layout Rule(s) to be loaded at the beginning of the transaction. A segment specified via TSEGS will be kept in the Input Transaction Rule regardless of whether there

are any displayable fields. The maximum combined number of segments that can be specified in the TSEGS and DBPATH operands is 100.

[TWINS=(segid,...)]

The TWIN operand specifies the list of twin segments to process on this generator statement. The first twin must be included in the list and it should also be entered in the DBPATH. All the twins must have the same parent segment ID, the same segment name, PCB Number, Length and field layout.

UPDATE= {NO | YES}

The UPDATE operand specifies that the generated Input Transaction Rule IDs should be included in the existing Secondary Option Menu Rule. A STEPLIB DD statement must point to the Rules Load library so that the existing SOMR may be retrieved. If UPDATE=YES and a SOMR cannot be found, the found, the generation will be terminated with an error condition. If UPDATE=NO is specified, the SOMR will be replaced. The default is YES if there is an existing SOMR, and NO if a SOMR is not found.

USRLANG=x

The USRLANG keyword on the GENERATE statement defines the language to be associated with that SYSID. If USRLANG= is not specified on either the SYSTEM or GENERATE statements, the Rules Generator will use the default language specified at install time. If USRLANG= is specified, the Rules Generator will load the specified MFC1G40 if it is not already in storage and will use that module to build rules and screens for the end user. The following USRLANG values are allocated to the corresponding language:

USRLANG Value	Language Module
E	English
F	French
G	German
J	Japanese
K	Korean
P	Portuguese
S	Spanish
W	Swedish

An example of this is:

```
GENERATE OPTIONS=CVALL,TRXID=T2,SYSID=MFC1,USRLANG=G
```

This example would cause the MFC1G40 module for German to be loaded and rules and screens would be generated for transaction T2 in German. Other USRLANG values of A-Z and 0-9 are available for other languages.

[VACANT=(segid,...)]

The VACANT operand is used to specify which twin aliases to reserve for insertion. The list must be a sub-set of the TWINS= list. No data will be retrieved by the online driver into these Segment ID's. The user will be able to enter keys and data to cause

insertions. The first segment named in the TWINS operand must not be specified in the VACANT list.

[WTOMSG='1 to 60 chars'] **(Batch only)**

The WTOMSG operand may be specified for the generation of a Batch Driver link-edit (OPTIONS=BDLE). The message text will be included in the generation of a Batch Driver Rule as a WTO message. A Batch Driver Rule is automatically assembled and included in the requested batch transaction driver load module and as such will not be stored as a separate member of a rule library. The WTO message will be displayed to the OS/VS system operator when execution of the batch transaction driver begins. A message text of 60 characters maximum may be specified with a WTOMSG operand. The WTOMSG operand may be repeated to enter a second 60 characters. One blank character will be generated following the last non-blank character of the first message text before the second message text is added. The default is that no message is sent.

[WTORMSG='1 to 60 chars'] **(Batch only)**

The WTORMSG operand may be specified for the generation of a batch transaction driver link-edit (OPTIONS=BDLE). The message text will be included in the generation of a Batch Driver Rule as a WTOR message. The WTOR message will be displayed to the OS/VS system operator with the reply option available to that operator for communication with the batch transaction driver. The WTOR's only function is to allow the system operator to stop the execution of the batch transaction driver. Prior to processing the next transaction message from the OS/VS sequential file, the batch transaction driver interrogates the Event Control Block (ECB) associated with the issued WTOR. If the ECB has been posted and the operator response was 'STOP', the batch transaction driver continues processing with the next transaction message. Note that the message must indicate to the operator that is the only valid reply. A message text of 60 characters maximum may be specified with a WTORMSG operand. The WTORMSG operand may be repeated to enter a second 60 characters. One blank character will be generated following the last non-blank character of the first message text before the second message text is added. The default is that no message is sent.

The GENERATE operands DECKS, ASMLIST, MAXKEY, and LRULE are a subset of those specified for the SYSTEM statement. Refer to the SYSTEM statement for their descriptions.

PRELOAD RULE

For IMS/VS execution, all static rules except composite rules load modules created by the Rules Generator may be preloaded into an application program region during the initial scheduling of an IMSADF II transaction. The preload is controlled by a resident common module, which will satisfy all subsequent requests for the rules. The loading of the rules is controlled by a Preload Rule, which lists all rules required to be loaded. The Preload Rule is generated through an OPTIONS=PREL operand followed by a series of parameters which specify rule IDs to be included. The IDs, specified as either segid or trxid parameter, along with the SYSID on the SYSTEM statement, will create rule names. Project/Group may also be specified on the GENERATE statement to name Input Transaction Rules. The operands which may appear on the preload GENERATE statement are in alphabetical order:

[ASRTABLE=(auditgrp,...)]

The ASRTABLE operand specifies the Audit static rules which will be loaded through the Preload Rule. Each auditgrp entry specifies a four-character Audit Group code or a rule name having more than four characters. If exactly four characters are specified, the SYSID is appended in front of auditgrp to form the rule name. If more than four characters are specified (eight character maximum), then auditgrp alone forms the rule name.

[DASRTAB=(auditgrp,...)]

The DASRTAB operand is used to specify audit static rule groups for deletion from an existing Preload rule. The input values are four to eight character audit group codes.

[DELETE=(segid,...trxid,...)]

The DELETE operand specifies that all rules containing the specified segid or trxid should be eliminated from the Preload Rule tables.

[ITTABLE=(trxid,...)]

The ITTABLE operand specifies the Conversational Input Transaction rules which will be loaded through the Preload Rule. Each trxid entry specifies a two-character transaction ID for which an Input Transaction Rule exists.

[MAPTABLE=(segid,...)]

The MAPTABLE operand specifies the Mapping rules which will be loaded through the Preload Rule. Each segid entry specifies a Mapping segment ID for which a Mapping Rule exists.

[OFRTABLE=(segid,...)]

The OFRTABLE operand specifies the Output Format rules which will be loaded through the Preload Rule. Each segid entry specifies an Output segment ID for which an Output Format Rule exists. If the segid specifies exactly two characters it is used as the output segment ID. If more than two characters are specified (eight character maximum), then it is used as the name of the Output Format Rule.

PREPOM= {<u>NO</u> | YES}

The PREPOM operand specifies that the Primary Option Menu Rule should be preloaded. The default is NO.

PRESOM= {<u>NO</u> | YES}

The PRESOM operand specifies that the Secondary Option Menu Rule(s) should be preloaded. The default is NO.

[SHTABLE=(segid,...)]

The SHTABLE operand specifies the Segment Handler Rule(s) which will be loaded through the Preload Rule. Each segid entry specifies a segment ID for which a Segment Handler Rule exists.

[SLRTABLE=(segid,...)]

The SLRTABLE operand specifies the Segment Layout Rule(s) which will be loaded through the Preload Rule. Each segid entry specifies a segment ID for which a Segment Layout Rule exists.

[SPTABLE=(trxid,...)]

The SPTABLE operand specifies the Special Processing Input Transaction rules which will be loaded through the Preload Rule. Each trxid entry specifies a transaction ID for which an Input Transaction Rule exists.

[TPITTABL=(trxid,...)]

The TPITTABL operand specifies the Nonconversational Input Transaction rules which will be loaded through the Preload Rule. Each trxid entry specifies a two-character transaction ID for which an Input Transaction Rule exists.

UPDATE= {**NO** | **YES**}

The UPDATE=YES operand value specifies that the generated Preload Rule should include existing Preload rule names. A STEPLIB DD statement must point to the Rules Load library so that the existing Preload Rule may be retrieved. If UPDATE=YES and a Preload Rule cannot be found, the generation will be terminated with an error condition. If UPDATE=NO is specified, the Preload Rule will be replaced. The default is YES if there is an existing Preload Rule, and NO if a Preload Rule is not found.

Each operand creates a table within the Preload Rule with an entry for each parameter. Depending upon which table is created, the entry in the table corresponds to the name of the rule that was generated according to IMSADF II naming conventions. For example, the Segment Layout Rule table (SLRTABLE) generates names composed of the application system ID (SYSID operand on the SYSTEM statement), constant 'SR' and the segid. Each Preload Rule generation will include the existing Preload Rule, if any, with the new entries or deletions.

SCREEN IMAGE INPUT

An alternative method of creating segment display screens is available through a function of the Rules Generator called Screen Image Input. This method allows the user to explicitly describe the screen layout by preparing a set of screen image statements that represent lines on a 3270 display device. These screen image statements immediately follow the GENERATE statement to which they apply. In addition, that GENERATE statement must specify SPOS=SIMAGE. The following discussion describes how screen image layouts can be prepared.

Notes for CICS BMS:

1. Row 1 column 1 through 14 are reserved for IMSADF II.
2. Multiple-page screen images can display up to a maximum of 255 segment fields. The actual maximum number of fields depends upon the number of segments specified in the Rules Generator run.

Screen Image Input for Conversational Processing

When the screen image format is used to generate the Segment Display screen, the following operands are not used:

Statement	Operands
SEGMENT	DISPLAY, MODE
FIELD	DISPLAY, MODE, SNAME, SLENGTH, SDECIMAL, SROW, SCOL
GENERATE	TRXNAME, SFORMAT, SHEADING

Literal data is entered directly in the screen image statements at the required screen locations. The existence of a field ID in the screen image input sets the DISPLAY operand value of that field to YES for that particular Segment Display screen generation. A screen image mode specification is used in place of the MODE operand of the FIELD statement for each field ID specified.

The column location of the first character of any text string in a screen image statement becomes the column position in the 3270 image. If a quote precedes a text string, that string will appear in high intensity and is broken with two or more blanks. The & (ampersand) character may be specified anywhere in a screen statement to represent the location of a variable field in the screen image. The & designates the location of the attribute byte. A screen image control statement must be separated by one blank character if mode is specified for the field. The only exception is symbolic names specified on SIMAGE (i.e., &S&G&A&B etc.). The format of a variable field may be one of the following:

&mfffff	for application fields
&mfffff.ss	for application fields with non-unique names

&adfname	for system fields
&xxxxxx	for symbolic reference to the tabular transaction field input
&control	for additional screen image control

where:

m — specifies the input mode for the field. A value must be specified for each field defined in the screen image. Values that may be entered are as follows:

4 modifiable (for all transaction modes)
5 modifiable
6 non-modifiable
7 modifiable non-display

ffff — specifies a one- to four-character field ID of a field to be included in the transaction defined by the screen image method. This ID must match the field ID of a previously specified FIELD statement in one of the segments defined as part of the input transaction with which this screen will interact. Segments are defined automatically as part of a DBPATH, or may be specified by the TSEGS operand of the GENERATE statement to be part of the Input Transaction Rule. The SLENGTH parameter specified on the FIELD statement will define the screen length for the screen image. Note that if SLENGTH is not specified, a default length is computed via a set of formulas that are dependent on the field type. A determination of whether fields overlap will be made for each field in the screen image based on the starting position and the SLENGTH for that field.

ss — specifies a two-character segment ID of the field to be included in the transaction. This ID is used to specify the correct segment definitions in cases where non-unique field names are present in transaction segments. If this ID is not specified, the correct segment is determined by first field ID found in scanning the data base path from the target segment back to the root segment. If a Special Processing transaction is being defined, the pseudo segment definition is scanned before the DBPATH segments. If TSEGS segment definitions are included in either Standard or Special Processing transactions, those definitions will be scanned before the Special Processing pseudo segment and the data base path segment definitions.

adfname — specifies that a keyword is to be placed in this location in the screen image. A keyword is defined as an IMSADF II system control field or an MFS system output literal. The following names may be specified: (The CURSOR operand may refer to these fields for cursor positioning.)

OPTION — return option field. A one-character field where the end user places the processing option such as N, C, or E. Required.

SYSMSG — 70-character system message field. This field will be displayed in high intensity and protected. Required.

ACTION — three-character screen action field. The ACTION field is required on multiple-physical-page Conversational Segment Display Screens. For conversational processing, IMSADF II will include the ACTION field on each page of the screen image if the keyword is omitted. This field provides an area into which the operator can input a paging request. Refer to the IMS Application Development Facility II Version 2 Release 2 User Reference manual for more information on multiple display screens.

TRAN transaction mode and TRXID. Format will be mss, where m is the current mode number chosen for the transaction and ss is the TRXID.

KEY the output value is the concatenated key of the hierarchical path being displayed. A user modification will result in a new path being displayed. Selection of the N option will display the next target segment, and the KEY value will be modified automatically. Required. Default length is 50 bytes. Length may be specified by MAXKEY operand of the GENERATE statement.

MODE eight-character transaction mode name. This name is equivalent to transaction modes 1-6 (INITIATE, ADD, RETRIEVE, etc.). Optional.

LTMSG 14-character message waiting. Optional. Ignored for CICS BMS.

LTNAME eight-character terminal identifier. Optional. Ignored for CICS BMS.

TIME eight-character format HH:MM:SS (current time). Optional. Ignored for CICS BMS.

DATE1 six-character format YY.DDD (current date). Optional. Ignored for CICS BMS.

DATE2 eight-character format MM/DD/YY (current date). Optional. Ignored for CICS BMS.

DATE3 eight-character format DD/MM/YY (current date). Optional. Ignored for CICS BMS.

DATE4 eight-character format YY/MM/DD (current date). Optional. Ignored for CICS BMS.

&xxxxxx Specifies a symbolic reference name that is present in the table following the screen image definition. One to six characters may be used. See VCOL= symbolic name below. Underscore characters may be used to specify SLEN for the field if the desired length is longer than the reference name. Note that underscores may also be used following the field id (eg. &5AA___.)

control specifies additional screen image control parameters. In each case the ampersand must be entered in column 1 and must be immediately followed by the control character. If control fields are used where two card images are required for each screen/printer layout line, for each screen/printer layout line, (DEVTYPE=3 or OPTIONS=OMFS,PRINTER=), a dummy card image must follow the card image containing the control field, i.e., two card images must still be processed. The following parameters may be specified:

&*

specifies that this is a comment statement. Comments follow the *.

&=nn

specifies the number of blank lines to insert on the screen before the next line is created.

&=P

specifies that the following screen image line will be placed on a new physical page.

&:

designates a tabular input which will specify a row and column in which a field is placed on the screen. Since five to eight characters are needed to specify a field location through screen image (mffff or mffff.ss), a series of short fields might be too widely spaced. If so &: can specify a row, column, and length for the display of the field. In addition, Audit parameters which apply only to this transaction can be specified. This eliminates the need to modify field statements for a requirement of a transaction. The tabular layout is fixed format, as shown below. The &* parameter may be used to create a heading for easier entry.

```
&*         1     1     2      3      3       4    5   5
&*---------2-----8-----4------1------8-------6----1---5--
&*   ID   *SLEN *VROW *VCOL  *VMODE *ASTATUS*KSEL*CLR*XHL
&: FFFF     6     3     25     5      PA      K    B   D
```

An operand value may be started in any column within the range specified and must be completed within the same range.

&ENDS

The &ENDS keyword is the trailer statement that ends each screen source.

ID={ffff / ffff.ss / name}

specifies the field name either by field ID and segment ID or DBD name. (Columns 3-10)

SLEN=nnnn

specifies the length of the display field (Columns 12-16). The length of the display field may also be specified thru the use of trailing underscore characters in the screen image. For example:

```
&5A___
&BB___
```

can be used to set SLEN for the specified fields to 5 bytes. If underscores are used in the screen image, the tabular SLEN for that field will be overridden.

VROW=nn

specifies the row of the display field. (Columns 18-22)

VCOL=nn

specifies the column of the display field. (Columns 24-29)

VCOL=symbolic name

A symbolic reference name may be used in place of row and column entries. The reference name (one to six characters with first character alpha) 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. YY-----&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 (i.e., &S&G&AA&BB etc.). Note that reference field usage takes precedence over ADFNAME usage. If &A is used for an application field, then &A could not be used for the IMSADF II action field. Instead, &AC or &ACT etc. would have to be used for the action field.

VMODE= {4 / 5 / 6 / 7}

specifies the mode of the display field. (Columns 31-36)

ASTATUS=FRPA

specifies Audit parameters which 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. (Columns 38-44)

F = FAUDIT=YES
R = REQUIRED=YES
P = PREAUDIT=YES
A = AFA=YES

Refer to the FIELD operands for a description of these parameters. If the ASTATUS transaction override columns are used, the transaction audit functions will be set to YES or NO. Including the character code sets the function to YES. Not specifying the character code sets the function to NO. These parameters override the ASTATUS values on the FIELD statement.

KSELECT=

specifies the key selection phase options to be performed on this field for this transaction. (Columns 46-48)

K	Key field displayed without auditing
KA	Key field displayed with primary and secondary auditing
KN	Key field not displayed with primary auditing
KP	Key field displayed with primary auditing only
KS	Key field displayed with secondary auditing
N	Not a related field for this transaction
R	Related field without auditing
RS	Related field with secondary auditing

CLR = Color Attribute

B	Blue
R	Red
P	Pink
G	Green
T	Turquoise
Y	Yellow

```
        W    White

    XHL = Extended Highlighting Attribute

        D    Default (no highlighting)
        B    Blink
        R    Reverse Video
        U    Underscore

    If multiple physical screens are specified, the tabular
    parameters must be entered with the image layout
    statements to which they apply.  For example:

          screen image layout (page 1)
          .
          .
          .
        &: tabular parameters (page 1)
        &=P
          screen image layout (page 2)
          .
          .
          .
        &: tabular parameters (page 2)

        System fields may be specified on more than one
        physical page of the Segment Display Screen Image.
        The following system fields may be entered on more
        than one page:

        LTMSG   14-character format msg waiting Qx
        LTNAME  eight-character terminal identifier
        TIME    eight-character format hh/mm/ss (current time)
        DATE1   six-character format yy.ddd (current date)
        DATE2   eight-character format mm/dd/yy (current date)
        DATE3   eight-character format dd/mm/yy (current date)
        DATE4   eight-character format yy/mm/dd (current date)
```

The set of screen image statements defining a transaction <u>must</u> immediately follow each GENERATE statement with the SPOS=SIMAGE operand value. The first statement entered will generate screen line 1 and so on up to the maximum line number allowed for the devices specified using the DEVTYPE operand. Enter blank statements or &=nn to space the screen as desired. Columns 1 through 80 of each screen image statement are processed as part of the screen image. Generation comments (* in column one) and statement sequencing in columns 72 to 80 should not be used as they will be generated as screen image literals. Enter &ENDS in columns 1 to 5 of a card to terminate the screen image. If IMSADF II system fields &OPT, &TRAN, &KEY, and &SYSMSG are not specified in the screen image, they will be placed on the last two lines of the screen. This means that overlapping fields will be detected if application fields are specified in the last two lines.

The following is an example of the use of the screen image input for Conversational processing:

```
SYSTEM SYSID=ZZZZ,SOMTX=TX

  SEGMENT ID=L0,LENGTH=100,NAME=DATAS1,LEVEL=1
    FIELD ID=KEYF,LENGTH=3,KEY=YES,NAME=KEY1
    FIELD ID=F2,LENGTH=8,TYPE=PD,DEC=3
    FIELD ID=F3,LENGTH=44,POS=55
  SEGMENT ID=L1,LENGTH=160,NAME=DATAS2,PARENT=L0
    FIELD ID=KEYF,LENGTH=6,KEY=YES,NAME=KEY1
    FIELD ID=F2,LENGTH=5
    FIELD ID=F3,LENGTH=33
  SEGMENT ID=L2,LENGTH=28,NAME=DATAS3,PARENT=L0,KEYN=KEY1
    FIELD ID=LOC,LENGTH=22,KEY=YES
    FIELD ID=DEPT,LENGTH=4,KEY=YES
    FIELD ID=PROJ,LENGTH=6,KEY=YES
    FIELD ID=ORG,LENGTH=12
    FIELD ID=GRP,LENGTH=24
  SEGMENT ID=PS,TYPE=PS
    FIELD ID=CNT1,LENGTH=4,TYPE=BIN
```

```
            FIELD ID=CNT2,LENGTH=4,TYPE=BIN
            FIELD ID=CNT3,LENGTH=4,TYPE=BIN

GENERATE OPT=CVALL,DBPATH=L1,TSEGS=(PS,L2),TRXID=T1,
         TRXNAME='SOM HEADING',PGROUP=LL,DISP=PG,SPOS=SIMAGE

            S C R E E N   H E A D I N G   F O R

               P R O J E C T   G R O U P   L L

MESSAGE:&SYSMSG
OPTION:&OPT

    SELECTION:&6KEYF.L0  &6KEYF.L1
    LOCATION:&6LOC      DEPARTMENT:&6DEPT
    PROJECT:&6PROJ

    &5CNT1     'ENTER STARTING VALUE

        ENTER YOUR ORGANIZATION

        ORG#----&5ORG     GROUP--&5GRP

                COUNTS ACCUMULATED
                ------------------
            COUNTA=&6CNT2     COUNTB=&6CNT3

&ENDS
```

Screen Image Input for Nonconversational Processing

The screen image layout operand value (SIMAGE) must be entered for each TPALL generation request. The following GENERATE statement is an example of the operands for a display screen for the TPALL option:

```
GENERATE  TRXID=xx,SPOS=SIMAGE,MODNAME=mmmmmmmm,OPTIONS=TPALL

   MODE:&X          SCREEN IMAGE LITERALS FOR TRANSACTION

   FIELD LITERAL 1:&5FID1    FIELD LITERAL 2:&6FID2

&         1    +    2    +    3    +    4    +    5    +   6
&*        0         0         0         0         0        0
&ENDS
```

where:

xx = the transaction id

mmmmmmmm = any one- to eight-character name. (Refer to the description of the MODNAME operand.)

The following IMSADF II system control fields may be specified:

- The transaction mode execution field, &X, positions the transaction mode field for Nonconversational processing. If &X is not specified in the screen image, the next to last screen line is generated as TRX:&X.
- An eight-character transaction mode may also be displayed on the Segment Display Screen by specifying &MODE. If &MODE is not specified in the screen image, only the numeric transaction mode will be displayed.
- The system message field, &SYSMSG, positions the SPAERMSG field in the screen image. If &SYSMSG is not specified, the last screen line is used for SPAERMSG.
- The screen action field &ACTION provides an area into which the operator can put a paging request. If not specified, the action keyword and input field will be placed on the second to last screen line.

LTMSG, LTNAME, TIME, DATE1, DATE2, DATE3 and DATE4 are MFS system literal fields that may be specified as part of a Nonconversational screen image layout. (See the preceding discussion of Screen Image Input for Conversational processing for a description of these fields.) These keywords do not apply to CICS BMS.

The placement of I/O fields within the screen image layout is identical to Conversational processing layouts.

AUDIT SPECIFICATION IN TABULAR FORM FOR BATCH INPUT TRANSACTION RULE

The Screen Image Tabular form may be used to create a Batch ITR using audit specifications. This allows you to define field definitions without audit specifications and allows you to put audit parameters in SIMAGE for the transactions you wish rather than on the field definitions themselves. This gives more flexibility when building input source to the Rules Generator. The same field definitions may be used to run either Conversational, Nonconversational, or Batch Processing without having to define audit specifications on the FIELD statements themselves. The following GENERATE statement is an example of operands for building a Batch Input Transaction Rule with audit specifications defined in the SIMAGE Tabular Input:

```
    GENERATE OPTION=BAIT,TRXID=xx,SPOS=SIMAGE

&*          1     1     2      3      3       4    5   5
&*---------2-----8-----4------1------8-------6----1---5--
&*   ID    *SLEN *VROW *VCOL  *VMODE *ASTATUS*KSEL*CLR*XHL
&:FFFF     :     :     :      :      : PA    :    :   :
&ENDS
```

where:

xx = the transaction id

The example would build the Batch Input Transaction Rule with audit specifications for PAUDIT=YES and AFA=YES. See the definition for ASTATUS under the Field Statement for all the valid Audit specifications.

INCLUDE STATEMENT

You may establish two application development libraries for storing data as common specification to one or more application transactions. A symbolic library (ADFLIB) is useful as a storage medium for 80-character statements. Usually, 72 characters of data and eight characters of identifier sequence numbers are maintained. A screen image library (IMAGELIB - usually containing 80-character records) may be installed in addition to the symbolic library. A screen image library containing 132-character records is also supported. (See GENERATE statement, DEVTYPE operand.)

The symbolic library may be used to include data base segment definitions in either IMSADF II or IMS/DBD formats. Pseudo, mapping, or output segment definitions in the IMSADF II formats may also be included. INCLUDE statements may not be a part of the member that is included (nested INCLUDES are not supported).

Generation requests for transaction driver link-edits, screen image layouts, and static rules may also be added to the symbolic library for inclusion into the Rules Generator input stream, when desired. If screen image statements are stored, statement sequencing must not be performed for that member since the sequence numbers will be interpreted as screen literals. The ADFLIB DD statement defines the data set name of the symbolic library when data is being included. Data is entered into members of a partitioned data set by using the IEBUPDTE VS/Utility.

The format of the INCLUDE statement is as follows:

```
INCLUDE MEMBERS=(memberl,member2,...membern)
```

or

```
INCLUDE SEGMENTS=(segidl,segid2,...segidn)
```

where

memberl,member2... are the names of the members of the symbolic library whose data is to be included. Data will be included in the order of the members specified.

segidl,segid2... are two-character segment identifiers of the segment definitions to be included from the symbolic library. A member name to be located in the symbolic library will be constructed from the current SYSID and SEGID, ssssxx.

Screen image input may also be retrieved from the screen image library (IMAGELIB) through an IMAGE= member name operand on a GENERATE statement. This causes the screen image definitions to be included after the GENERATE statement in the Rules Generator input stream. Only one member can be included by an IMAGE operand. A GENERATE statement with an IMAGE operand may not be used as a part of a member of an INCLUDE statement.

RULE STATEMENT

The RULE statement provides control information to the Rules Generator for entering Assembler language rule source that is to be assembled and link-edited in a specified rules load library. The rule source option (DECKS=YES) is available for obtaining an Assembler language source deck for any rule. For example, the developer may obtain an Input Transaction Rule source for the purpose of adding a secondary transaction ITRMOD statement to the output of the Rules Generator.

The RULE statement is generated automatically as part of the output deck by the Rules Generator when DECKS=YES is specified. It is a header statement that precedes the generated rule source, and it identifies the rule name, the Linkage Editor control statement option, and the Assembler listing option. The delimiter, &ENDR, follows the generated rule source. The RULE statement and &ENDR delimiter are generated by the Rules Generator for each rule source deck created.

Multiple sets of rule source statements can be processed by the Rules Generator. Each set must have a RULE statement and an &ENDR delimiter.

An example follows of a Standard Secondary Option Menu Rule generated by the Rules Generator when DECKS=YES:

```
RULE  MEMBER=SAMPSMSS,LOPT=2,SSI=0000AD08
SAMPSMSS  SMRULE  ID=PA,IMSTX=OR,KEYSL=YES,DESCR='PART SEGMENT',
                  TYPE=STD
          SMRULE  ID=IV,IMSTX=OR,KEYSL=YES,DESCR='INVENTORY',
                  TYPE=STD,LAST=YES
          END
&ENDR
```

A summary of the RULE statement operands is shown in Figure 2-20 on page 2-110.

```
Name      Statement   Operands      Values
          RULE        MEMBER=       mmmmmmmm

                      ,LOPT=        { 1 }
                                    { 2 }
                                    { 3 }
                                    { 4 }
                                    { 5 }

                      ,ASMLIST=     { LIST   }
                                    { NOGEN  }
                                    { NOLIST }

                      ,LANGUAGE=    { COBOL  }
                                    { ASMINT }
                                    { ASMSTD }
                                    { PL/I   }

                      ,SSI=         xxxxxxxx

                      ,LINKLIB=     { SYSLMOD }
                                    { xxxxxxx }

                     [,AEXIT=       xxxxxxxx]

                      ,ALANG=       { COBOL  }
                                    { ASMINT }
                                    { ASMSTD }
                                    { PL/I   }

                     [,DEXIT=       xxxxxxxx]

                      ,DLANG=       { COBOL  }
                                    { ASMINT }
                                    { ASMSTD }
                                    { PL/I   }

  &ENDR
```

Figure 2-20. RULE Statement Summary

OPERANDS (in alphabetical order):

[AEXIT=xxxxxxxx]

The AEXIT operand names a user-written Audit exit which is to be included in a transaction driver link-edit. The object module named by xxxxxxxx must exist in the data set pointed to by OBJLIB.

ALANG= { **COBOL** / **ASMINT** / **ASMSTD** / **PL/I** }

The ALANG operand specifies the language in which the Audit exit is written. The default is COBOL.

ASMLIST= { **LIST** / **NOGEN** / **NOLIST** }

The ASMLIST operand is described in the SYSTEM statement section.

[DEXIT=xxxxxxxx]

The DEXIT operand value names a user-written DL/I exit routine which is to be included in a transaction driver link-edit. The object module named by xxxxxxxx must exist in the data set pointed to by OBJLIB.

DLANG= { **COBOL** / **ASMINT** / **ASMSTD** / **PL/I** }

The DLANG operand specifies the language in which the DL/I routine was written. The default is COBOL.

LANGUAGE= { **COBOL** / **ASMINT** / **ASMSTD** / **PL/I** }

The LANGUAGE operand is used to specify whether a language interface routine is required as part of the link-edit. If PL/I is specified, a PL/I language interface module will be included in the link-edit. The other options do not require an interface module. The default value is COBOL.

LINKLIB= { **SYSLMOD** / **xxxxxxxx** }

The LINKLIB operand specifies the name of a data set which will receive the output of the rules generation or link-edit process. The default is SYSLMOD.

LOPT= { **1** / **2** / **3** / **4** / **5** }

The LOPT operand specifies the Linkage Editor control statement option. Value 1 indicates that a segment handler is to be link-edited. Value 2 is the default and indicates that a rule is to be link-edited. Value 3 specifies that a conversational Special Processing load module link-edit is to be performed. Value 4 specifies that a nonconversational Special Processing load module link-edit is to be performed. Value 5 specifies that a batch transaction driver load module link-edit is to be performed.

MEMBER=mmmmmmmm

The MEMBER operand specifies the one- to eight-character rule name. The rule name will be the rule member name in the rules load library.

SSI=XXXXXXXX

The SSI operand specifies hexadecimal information which will be used in a Linkage Editor SETSSI statement to set the system status index of the directory entry when a rule or transaction driver load module is link-edited. Eight hexadecimal characters are required if this operand is used. Refer to the Operating System Linkage Editor manual for additional information on the SETSSI command.

DELIMITER: &ENDR

The &ENDR delimiter is the trailer statement that ends each rule source. The delimiter must occur in positions 1 to 5 of the input record.

RULES GENERATOR CONTROL STATEMENT EXAMPLES

This section demonstrates how to install the Sample Problem that is distributed with IMSADF II. Refer to Appendix A, IMS Application Development Facility II Version 2 Release 2 Application Development Guide for a complete listing of the Rules Generator input. Before writing the Rules Generator SYSTEM statement, you must choose a four-character major application system ID, a two-character data base ID, a Project/Group code for sign-on authorization, and a cluster code. In this case, 'SAMP', 'PA', 'ZZ' and 'OR' characters are chosen as IMSADF II control characters. In addition, 'S A M P L E P R O B L E M' is chosen as the major sample application screen heading.

The cluster code chosen is included as part of the link-edited name of the transaction driver module. The IMS/VS system generation must include this name as a transaction and the developer must perform appropriate PSB generations to provide the data bases that will be processed with the cluster code that was chosen.

Since this is a new application, a Sign-on screen and a Primary Option Menu must be generated. The SYSTEM statement and first GENERATE statement for this application are specified as follows:

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

This application is composed of Standard Processing transactions which update the various segments in the data base and also a Special Processing transaction which performs inventory calculations and update. You must define all data base segments in the hierarchical paths of the data base. Each segment definition will contain a SEGMENT statement and FIELD statements to describe the fields within the segment.

Each segment is assigned a two-character segment ID (PA, PD, IV, CY) that is unique within the application system. The DBD segment type name does not use IMSADF II naming conventions, so each SEGMENT statement will have to specify the actual name. This also will be true of each sequence field defined in the DBD generation.

The root segment has no parent and must be specified as a LEVEL=1 or PARENT=0 segment. The Rules Generator statements to define the root segment are as follows:

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

  FIELD   ID=KEY,LENGTH=17,KEY=YES,DISP=YES,SNAME='PART NUMBER'

  FIELD   ID=DESC,POS=27,LENGTH=20,DISP=YES,SNAME='DESCRIPTION'
```

Note that, through the specification of the DISPLAY operand, the root segment fields also are to be part of any segment display screens for

segments in the data base. Since the MODE operand is not specified for these fields, the update capability will be generated by default. If the root segment has its own Input Transaction Rule, then both fields can be updated. When the root segment is part of the INVENTORY segment display, the root key field will not be modifiable, but the DESC field will be modifiable (MODE=5).

The INVENTORY segment is defined as follows. In this example, the DBD sequence field is subdivided into its components, AREA, PROJECT, and DIVISION. The view of the INVENTORY segment then will have more fields than the DBD description. However, the segment I/O performed by IMSADF II must refer to the sequence portion of the data base segment as a single field. Each component field must be specified with the KEY=YES attribute. The SEGMENT statement must contain the DBD name of the combined sequence area in addition to the segment type name. Data from several fields within the segment on the Secondary Key Selection screen are to be provided to help the end user select the proper inventory segment data for retrieval. To accomplish this, supply a secondary key select heading. In most cases, an automatic heading generation would be acceptable. But for this case, the available 72 columns would be used up by the multiple key screen names. A key selection requirement is that all key field data must appear on the Secondary Key Selection screen. The developer has selected UNIT PRICE, CURRENT REQUIREMENTS, ON ORDER, TOTAL STOCK, and the PLANNED and UNPLANNED DISBURSEMENT fields to be data that is related to the selection of the inventory occurrence. Since the fields are defined as zoned decimal in the data base, one extra column for the sign and one for the decimal point must be taken into account.

The Rules Generator statements are as follows:

```
SEGMENT ID=IV,PARENT=PA,NAME=STOKSTAT,KEYNAME=STOCKEY,LENGTH=160,
        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='IV SEGMENT CODE',DISP=NO,COL=1,SLENGTH=2

      .     .
      .     .
      .     .
```

You next must choose which rules and screens are required for the specific segments just defined. Since each segment is processed as a Standard processing transaction, a complete set of rules is required for each segment definition. A Secondary Option Menu Rule will be requested. This rule will contain a transaction ID for each Input Transaction Rule generated. Finally, a Standard Processing transaction driver must be link-edited using the cluster code as the program ID (PGMID).

```
GENERATE OPTIONS=SGALL
GENERATE OPTIONS=CVALL,TRXID=PA,TRXNAME='PART SEGMENT'
GENERATE OPTIONS=CVALL,TRXID=PD,TRXNAME='STANDARD INFORMATION'
GENERATE OPTIONS=CVALL,TRXID=IV,TRXNAME='INVENTORY'
GENERATE OPTIONS=CVALL,TRXID=CY,TRXNAME='CYCLE COUNT'
GENERATE OPTIONS=SOM
GENERATE OPTIONS=STLE,PGMID=OR
```

To provide additional control of the depletion of inventory stores, the Special Processing option of the application system is used. In this case, a new set of counter fields is defined along with a PLANNED or UNPLANNED notification field. In order to process these fields and return them to the display screen, you must have one or more mapping rules available. For this example, since all fields mapped in are to be mapped at the end of each processing sequence, one mapping rule is sufficient. The following statements are then provided to define the Special Processing application segments to the Rules Generator:

```
SEGMENT ID=CD,TYPE=PS,SNAME='CLOSE/DISBURSE INVENTORY',DISP=YES
  FIELD ID=CLOR,L=7,TYPE=DEC,SNAME='CLOSE ORDER',SL=7
  FIELD ID=CLST,L=7,TYPE=DEC,SNAME='STOCK INCR',SL=7
  FIELD ID=DQTY,L=7,TYPE=DEC,SNAME='DISBURSE QTY',SL=7
```

```
      FIELD ID=DIPU,L=1,AUDIT=YES,SNAME='PLANNED/UNPLAN'

    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
```

You must then determine what rules are necessary for the Special Processing Routine application. First, provide the basic Segment Layout Rules for any new segment definitions or data base definitions that were not in the Standard Processing paths.

The Segment Layout Rules for the Pseudo segment (CD) and Mapping segment (M1) are generated through OPTION=SEGL.

A Key Selection Rule has already been created for the IV path in the previous GENERATE statement. You will need access to the PD segment, which is a segment not in the path of the primary key selection target, i.e., the IV INVENTORY segment, but which is immediately retrievable from the key of the selected IV segment because there is only one occurrence of the PD segment in the data base record. Space will be reserved for the PD segment and the CD pseudo segment through the TSEG operand.

You must also specify the data base path that is to be included in the Input Transaction Rule and for which a Key Selection Rule is to be generated. A single DBPATH value indicates to the Rules Generator that the IV segment is the target segment in a data base path.

Following the above step, request a Special Processing Routine link-edit and include the mapping rule and Segment Handler IDs that will be referenced in the Special Processing Routine logic.

Finally, request that the Special Processing transaction ID be included in the Secondary Option Menu Rule.

The following generation requests were presented to the Rules Generator:

```
    GENERATE SEGMENT=(M1,CD),OPTIONS=SEGL
    GENERATE TRXID=CD,TSEG=(CD,PD),DBPATH=IV,OPTIONS=(CVALL),
             LANGUAGE=ASMINT,BYPASS=YES,SPECIAL=YES,KEYSL=YES
    GENERATE PGMID=CD,OPTIONS=SPLE,MAPTABLE=(M1),SHTABLE=(PD,IV),
             LANGUAGE=ASMINT
    GENERATE OPTIONS=SOMSS,UPDATE=YES
```

USING THE RULES GENERATOR EXECUTION PROCEDURE

The ????G procedure is a one job-step procedure that executes a Rules Generator main program. During the generation phase of processing, the Assembler and Linkage Editor are invoked repeatedly for rule creation and installation. Figure 2-21 shows the data definition names required for each phase of rules generation.

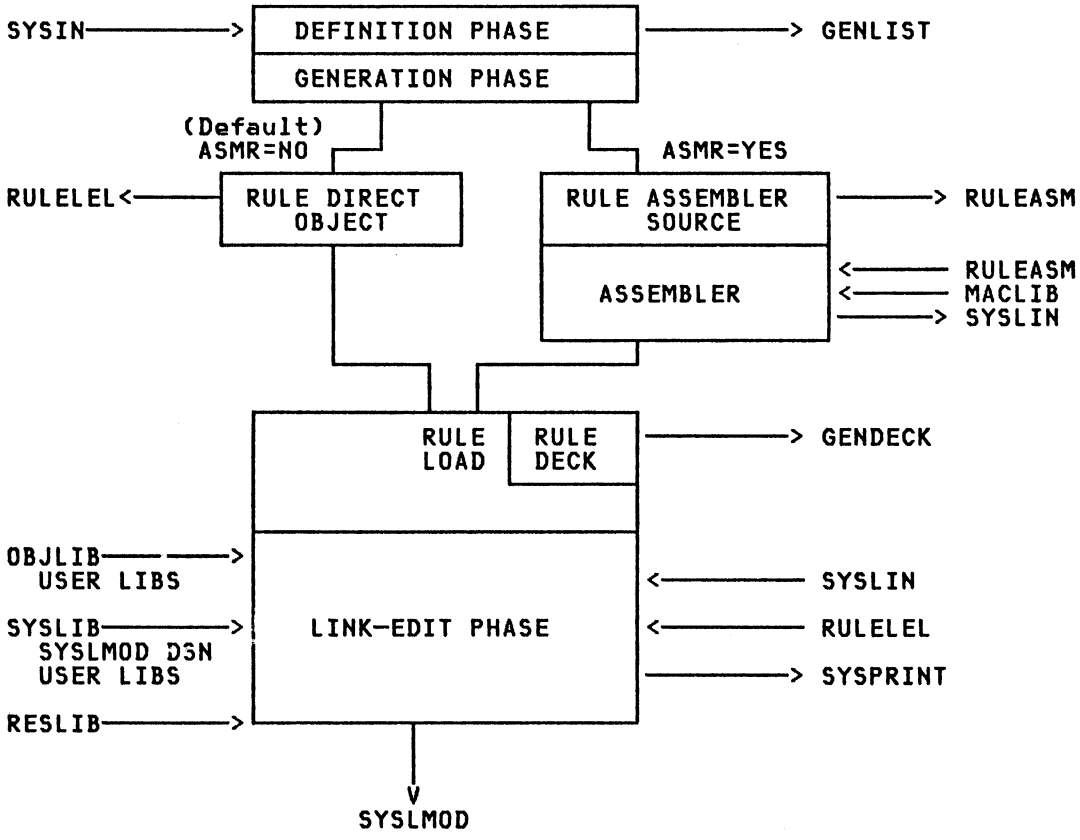


Figure 2-21. Rules Generator Data Definition JCL Requirements

The following JCL execute statement shows the usage of the MFCG procedure.

```
//STEP  EXEC  ????G  RULES=LIB1,PARM=(500,2,100,AAAA)
```

RULES is a symbolic parameter that specifies the prefix of the name of the data set where the load modules of the rules are to be placed. Suffix is '.RULES.LOAD'. The default value is TEST.

```
                 500            2            100
PARM=   #fields  ,   #screens     ,#columns          [,sysid]
```

The following section describes the use of the execute parameters for the Rules Generator:

OPERAND	DESCRIPTION
#fields	Maximum requirement for one SYSTEM statement will be the sum of the following: • number of field definitions in all segments. • number of Secondary Key Selection heading lines entered by the SKLEFT operand. • maximum number of fields in one TPALL transaction.
#screens	Maximum requirement for one GENERATE statement determined by the number of screen image pages for one transaction. The maximum number that may be specified is 22.
#columns	Maximum requirement for one SYSTEM statement will be the sum of the number of column definitions in all TABLE statements.
sysid	Four-character application system identification. The alpha operand value will be used to set this parameter. The identification specified will override the SYSID operand on any subsequent SYSTEM statements.

The following is a list of the DD statements in ????G and a description of their use. This list is ordered alphabetically.

DDNAME	DESCRIPTION
ADFLIB	Partitioned data set containing Rules Generator rule definitions
ADFSQLHO	The source code for generated Table Handler Rule
ADFSQLHW	The work file used by Table Handler Rule Generation
ASMLIST	SYSOUT data set for the assembly phase
CICSMAPS	The partitioned data set used for output of the CICS Basic Mapping Support (BMS) screen source defining the different mapsets for CICS. Each member may be input to the BMS Utility to produce the mapset load module.
CICSPPT	The sequential data set used to output the CICS Processing Program Table (PPT) entries for rules, mapsets, transaction drivers, and their aliases created in this Rules Generator run
DBRMLIB	The partitioned data set used by DB2 precompiler to store DB2 DBRM
DB2PRINT	The data set used by DB2 precompiler as SYSPRINT file
DB2TERM	The data set used by DB2 precompiler as SYSTERM file
DDADFX	The sequential data set which will contain the Rules Generator Extract data to be used by the Data Dictionary ADFIN Format Processing. If this data set is used, its LRECL must be stated as 256 bytes. **Note:** This ddname can be overridden using the DDADFX keyword on the OPTIONS=ADFX on the Generate Statement.
GENDECK	The output data set used to punch the rules source if the DECK option is chosen
GENLIST	SYSOUT data set for the Rules Generator, contains a list of the input statements and associated diagnostics

IMAGELIB	Partitioned data set containing screen image layouts
ISPF	The sequential data set which will contain the Rules Generator Extract data to be used by the IADF Table Build module. If this data set is used, its LRECL must be stated as 1200 bytes.
MACLIB	MACRO library for the assembly phase contains the IMSADF II Assembler macros for rules creation
PGMLOAD	PGMLOAD is used to route link-edit load modules to PROGRAM.LOAD rather than to RULES.LOAD in the install process
RULEASM	Intermediate data set used to pass the rules to the assembly phase of the Rules Generator
RULELEL	Intermediate data set used to pass the rules to the link-edit phase of the Rules Generator
SCREENS	The sequential data set used for output of the IMS/VS Message Format Services (MFS) screen source defining the different screen formats. This file may be input to the MFS Utility.
STEPLIB	Data set containing the Rules Generator Load Module and data set containing the SOMR(s) and Preload Rule
SYSIN	The control statement input to the Rules Generator
SYSLIB	The load module library for automatic library search for the Special Processing Routine link-edit The rules load library being used for SYSLMOD should be included as a shared library. The language product load libraries should also be included (PL/I or COBOL). The load module library containing Special Processing, Audit and DL/I exits should be included as well.
SYSLIN	The object module output from the assembly phase, which will be part of the primary input to the link-edit
SYSLMOD	The output data set(s) to contain the link-edit load modules. Specific output data set can be controlled by the LINKLIB operand on the GENERATE statement.
SYSPRINT	SYSOUT data set for the Linkage Editor
SYSUT1	Intermediate work data set used by the assembly phase and by the link-edit phase
SYSUT2	Intermediate work data sets used by the assembly phase
SYSUT3	Intermediate work data sets used by the assembly phase

RULES GENERATOR RETURN CODES

A job step for IMS/VS MFS execution can be placed after the Rules Generator job step for runs that will create MFS source. One of several return code values may be encountered during a Rules Generator execution step. A non-zero return code indicates that an error condition was found in the input data. The Rules Generator return codes and their meanings are shown below. The highest value found during the job step will be returned at the conclusion of all input processing. The following job steps should test for a non-zero return to prohibit processing on an error condition. If an error was encountered during the assembly of link-edit phases of the rules generation process, the Rules Generator will include the 'ASSEMBLY DIAGNOSTIC ENCOUNTERED' or 'LINK EDIT DIAGNOSTIC ENCOUNTERED' message immediately following the GENERATE statement that contained the request for the rule installation. The developer must review the ASMLIST or SYSPRINT system output listings for more descriptions of the user error.

The possible return codes are:

RETURN CODE	DESCRIPTION
0	NO ERRORS ENCOUNTERED IN ANY PHASE
4	MAPPING RULE VARIABLE NAME NOT FOUND
8	APPLICATION DEFINITION ERROR
12	SEVERE ASSEMBLER OR LINKAGE EDITOR ERROR
16	RULES GENERATOR DATA SETS COULDN'T BE OPENED
20	ASSEMBLER ABORT...CHECK REGION SIZE OR JCL

Figure 2-22 illustrates how the Rules Generator procedure can be executed with the IMS/VS MFS Utility using a temporary screen source data set.

```
//JOB
//          EXEC        ????G
//G1.SYSIN  DD          *
  SYSTEM                SYSID=SAMP,SHEADING='SAMPLE PROBLEM'
  GENERATE              OPTIONS=(SIGN,POM)
/*
//          EXEC        MFSUTL,COND.S1=(0,LT)
//S1.SYSIN  DD          DSN=&&MFS,UNIT=SYSDA.DISP=(OLD,DELETE)
```

Figure 2-22. Procedure Execution Using Temporary Screen Source Data Set

In this example, a Primary Option Menu Rule is stored in the TEST.RULES.LOAD library with the member name SAMPPM. This library name is the default for the procedure. In addition, a Sign-on screen source is created by the Rules Generator and passed to the IMS/VS MFS Utility to create MFS load modules. Note that each rule is stored in the rules load library with a unique name according to IMSADF II naming conventions.

If the screen source must be stored, Figure 2-23 is an example of the required JCL for the Rules Generator and IMS/VS MFS procedure execution using a partitioned data set member for screen source.

```
//JOB
//          EXEC        ????G
//G1.SCREENS DD         DSN=SAMP.SCREENS.MFS(SIGNON),
                      DISP=(OLD,PASS)
//G1.SYSIN  DD        *
     SYSTEM           SYSID=SAMP,
                      SHEADING='SAMPLE PROBLEM'
    GENERATE          OPTIONS=(SIGN,POM)
/*
//          EXEC      MFSUTL,COND.S1=(0,LT),SNODE='SAMP.SCREENS',
                      SOR=MFS,MBR=SIGNON
```

Figure 2-23. Procedure Execution Using Partitioned Data Set Member

The assembly and link-edit phases of the rules generation function can be performed in a second pass of the Rules Generator. Figure 2-24 illustrates how the assembly and link-edit phases can run as a separate step.

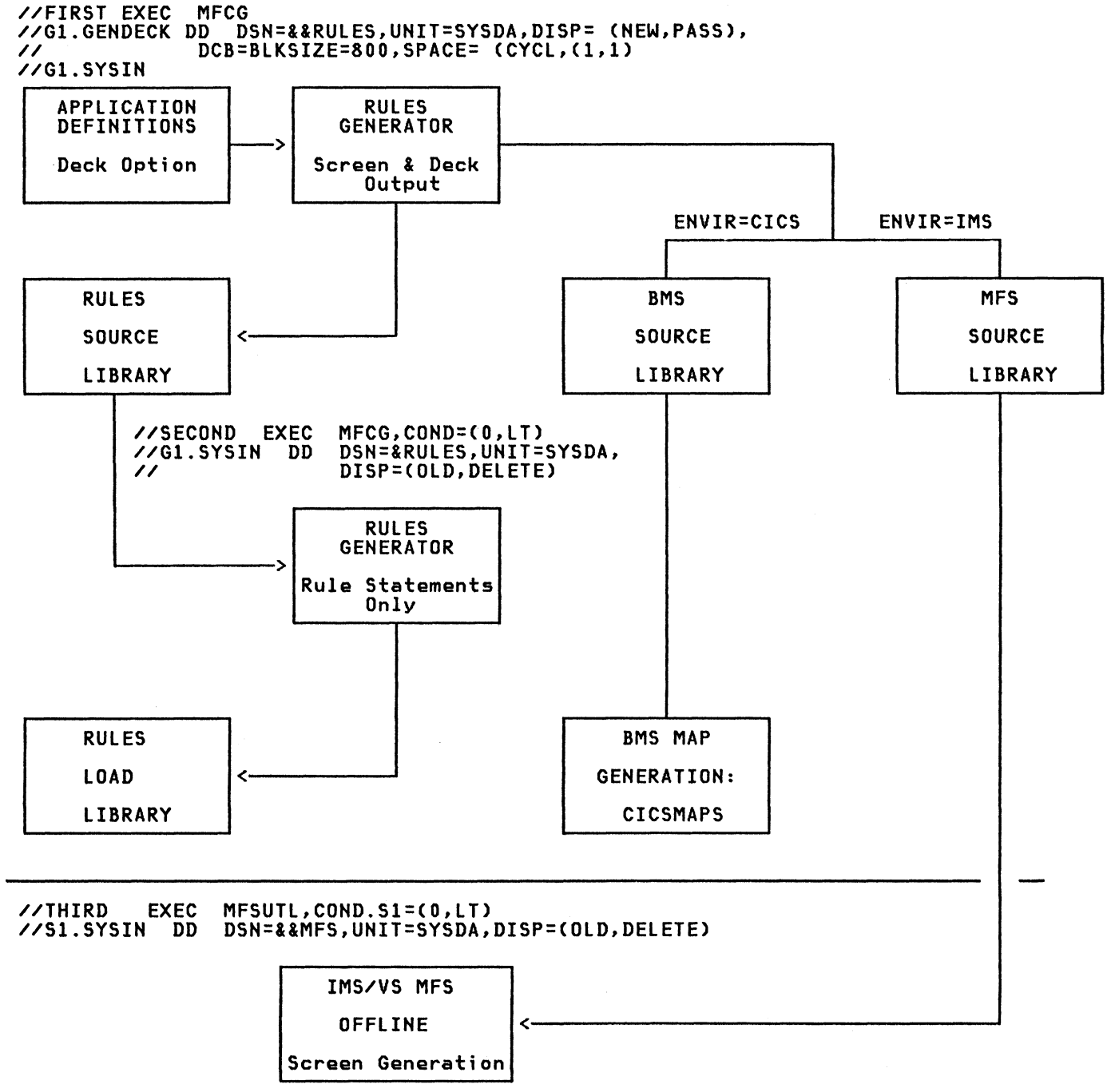


Figure 2-24. Rules Generator Usage at the Assembly Phase

The application definition control statements are submitted to the first execution of the Rules Generator with a DECKS=YES request. Assembler source members for the rules and screen source will be generated for the definitions supplied. A second Rules Generator execution step immediately following the first, assembles and link-edits each rule created in the first step. In this second step, SYSIN is defined as the data set containing the decks created in the first step. If a user

input error occurs in any definition or generation request in the first step, the second step will not be executed.

The Rules Generator will either generate MFS or BMS screens. If the parameter ENVIR specifies IMS (the default) the examples in Figure 2-22 and Figure 2-23 will occur. If ENVIR=CICS, Figure 2-25 is an example of the required JCL for the Rules Generator to create BMS maps and these maps are placed in the CICSMAPS data set.

```
//JOB
//          EXEC      ????G
//CICSMAPS   DD       DSN=SAMP.SCREENS.BMS(SIGNON),
                      DISP=(OLD,PASS)
//G1.SYSIN  DD        *
     SYSTEM           SYSID=SAMP,SHEADING='SAMPLE PROBLEM',
                      ENVIR=CICS
    GENERATE          OPTIONS=(SIGN,POM)
/*
```

Figure 2-25. Procedure Execution for Generating BMS Maps

CHAPTER 3. AUDIT LANGUAGE

IMSADF II supplies many built-in functions which control the retrieving, displaying, and updating of data. In addition, most applications require logic and editing capability. IMSADF II supplies some of these functions in precoded modules that are automatically executed at predefined points within the transaction. Additional support is provided through a procedural language called the Audit Language. This language allows the developer to define operations ranging from standard compares and data moves to data base calls, transactions switches and screen control. The language is designed so that the developer is not required to know the data base structures, DC system (IMS/VS or CICS/OS/VS) or screen attributes (MFS or BMS) under which the transaction is executing.

IMSADF II controls these logic requirements through rules. The rules (both static and dynamic) direct IMSADF II to execute appropriate precoded routines to perform the desired logic. The Audit language is used to define these rules.

Audit Language statements are translated into a set of operations in much the same manner that COBOL or PL/1 are translated into Assembler language. These operations are either stored in the Audit Data Base or compiled into an audit static rule. The format of these operations and the way in which the Auditor uses them is explained in Chapter 5, "Audit Logic Processing." In some cases it may be helpful to refer to Chapter 5, "Audit Logic Processing" when determining how a specific operation is performed.

AUDIT LANGUAGE PROCESSING

The Audit Language processor is a batch program. It reads 80-character Audit language source statements and generates:

- Rule output in the form of either:
  - IMSADF II batch transactions to build the records in the Audit Data Base, or
  - Audit static rule source statements to build a static audit load module.
- A compiled listing showing:
  - cross reference between input and generated output, and
  - diagnostic errors.

The generated batch transactions are used as input (TRANSIN) for creating Audit Data Base records. This procedure is described in "Updating Dynamic Rules Data Bases in Batch Mode." The first transaction for each audited field deletes all the entries for that audit record. The following records recreate the operation and data descriptors in the data base. This requires that all the Audit Language source for a field should be included whenever a generation is executed. Also Primary and Secondary key Audits should be combined. The manner in which they are combined is explained later under the topic "Definition Statements".

SYNTAX NOTATION USED IN THIS CHAPTER

In this chapter, the following syntax notation conventions are used:

1. Upper case letters and words appearing in a format description are entered exactly as they appear.

2. Lower case letters and words in a format description represent substitutable information.

3. The delimiters <> are never coded. They are used to indicate that the information they enclose is optional and the user may code one or more of the items (or none of them).

4. The delimiters () are used to indicate the grouping of literals (data descriptors) and keywords. Their meaning will be evident from their context. They are never coded but are used to indicate that, while the information they enclose is optional, at least one of the items must be selected. In a very few cases tha () have a different meaning. This will be noted for each particular case.

5. An ellipsis (...) is never coded. The periods may be horizontal or vertical and are used to indicate that the information preceding the ellipsis may be repeated.

AUDIT LANGUAGE SYNTAX

The following Syntax Conventions apply to using the Audit Language:

1. Each input image can contain data in columns 1 through 72. Columns 73 through 80 can be used for sequence numbering.

```
IF SACDCLOR = 'AB,CD','EF,GH'                    00000001
```

2. The input format is completely free form except within quotes. Extra blanks are ignored. Statements can be spanned across multiple images.

```
IF SACDCLOR = 'AB,CD',
    'EF,GH'
```

3. A blank delimiter is required between all elements of a statement.

```
ABCD EQ 10
ABCD =  10
ABCD =  FIELD1 * FIELD2
```

4. An asterisk in column 1 identifies the statement as a comment.

5. Alphanumeric literals can always be enclosed in single quotes. Two quotes within quotes will be translated to a single quote. All other characters are allowed within quotes. However, a comma and a right parenthesis are not valid characters for the Auditor but are treated as delimiters.

6. Numeric constants or numbers may include decimal points and a leading minus sign. Imbedded blanks are not allowed.

7. The audited field can be specified by the 1-4 character field ID.

8. Related fields must be specified with the fully qualified field name (ssxxffff).

DEFINING AN AUDIT ROUTINE

The following statements define the environment of the audit routine(s) being generated. These statements define the audit routines name (key), the audit call in which the routine is invoked, and the phase within the routine to be used. This information is required to produce the correct Audit data base keys and place the audit correctly in the data base or static rule.

ADFID Statement

ADFID = Four-character application ID of the IMSADF II batch application driver used to update the Audit Data Base. The application ID becomes the first four characters of the generated transaction output.

If ADFID is omitted, the application ID defaults to the ADFID specified in the DEFADF macro during installation. This should be used only if

the Audit Language generator is installed under a different ADFID than the batch driver used to update the Audit Data Base.

NAMING THE AUDITS

Each generated audit record has a 16-character key associated with it. At execution, the Auditor develops the appropriate key value depending upon the field being audited and the call which invoked the Auditor. Since different audits require different names it is appropriate at this time to describe the possible name formats. It may be helpful to refer to Chapter 5, "Audit Logic Processing" for a detailed discussion of audit calls (Primary key selection, Secondary key selection, Preaudit, Standard audit) and phases (Automatic Field Assignment, Field Audit/Logic, Message Sending)

The audit key names are constructed in one of five formats:

Format 1: Used to specify Audit records for Preaudit, Standard Audit and Key Audit (KANAME=ALT) calls and contains an eight-byte Audit group code plus an eight-byte fully qualified field name. The Audit group code is formatted as follows:

ssssyyyy where

ssss = Application System Id (SYSID)

yyyy = Audit group specification (AGROUP)

Since an Audit group code is associated with an Input Transaction Rule, a field can have different logic processing, depending upon which Input Transaction Rule is currently being processed.

The eight-byte field name consists of:

ssxxffff where

ss = Two high-order bytes of Application System ID

xx = ID of segment that contains the field

ffff = Field ID

This name matches the Segment Layout Rule field name created by Rules Generator input.

Format 2: Used to specify Primary and Secondary key audits (KANAME=STD) and is constructed as follows:

KEYAUDITssxxffff

The first eight bytes are a constant 'KEYAUDIT' followed by an eight-byte field name. Key Audits with this naming convention reside within a different data base record than Preaudits or Standard Audits for the same field.

Key Audits also can be constructed using format 1. The format 1 convention is used if KANAME=ALT is specified on the Input Transaction Rule. If KANAME=ALT is not specified format 2 is used. Refer to KANAME parameter in Chapter 2, "Rules Generator."

Format 3: Used to specify "common audits". An installation may have fields of the same name used in many segments and applications. An example might be a date field called 'DATE'. The logic processing for this field is constant for all occurrences. The user can designate the field name as a "common field" through Rules Generator input. This is done by specifying CAUDIT=Y each time the field is described to the Rules Generator. This will mark the field entry in the Segment Layout Rule as a Common field. When a field name is specified as common, only one set of logic need be created and maintained for all of its occurrences. The key for a common field has the following format:

COMMON000000ffff

The first twelve bytes are constant (COMMON000000) and bytes 13-16 contain the Field ID.

Format 4: Used to specify audit subroutines. An audit operation is available which allows branching to an audit subroutine from any other audit routine. In this operation, the user specifies the key of the subroutine. Any meaningful 16-byte definition may be used for this key. For example:

SUBROUTINE-#0001
or
0000000000000001

would both represent a valid key. If audit load modules are used, the first eight characters of this format are restricted to alphabetic or numeric characters and the first character must be alphabetic.

Format 5: Used to specify audit table entries. Any meaningful 16-byte definition may be used for the key. If audit load modules are used, the first eight characters of this format are restricted to alphabetic or numeric characters and the first character must be alphabetic.

Each of the five name formats is automatically generated through the use of one or more of the following definition statements.

DEFINING THE AUDITED FIELD

The following five statements are used to define audits using key formats 1, 2, and 3.

SYSID = Four-character majo. systems identification.

If a common audit routine is created, 'COMMON' should be substituted and the identifier statements AGROUP and SEGID should not be specified.

AGROUP = Four-character audit group name.

This name must coincide with the Audit group generated by the Rules Generator in the Input Transaction Rule. The default is YYYY.

SEGID = Two-character segment ID of the audited field.

FIELD = One- to four-character field ID of the audited field.

KANAME = (STD,ALT)

If STD, key audits are constructed with a key format 2 layout (KEYAUDITssxxffff). If ALT, key audits are with a key format 1 layout.

The following examples show how different audit names are created.

Standard name (Format 1)

```
SYSID   =   SAMP
AGROUP  =   MAUD
SEGID   =   CD
FIELD   =   CLOR
```

- will generate a key of -

SAMPMAUDSACDCLOR

Common audit (Format 3)

```
SYSID   =   COMMON
FIELD   =   CLOR
```

- will generate a key of -

COMMON000000CLOR

Keyaudit (Format 2)

```
SYSID   =   SAMP
AGROUP  =   MAUD
SEGID   =   CD
FIELD   =   CLOR
KANAME  =   STD
```

- will generate a key of -

KEYAUDITSACDCLOR

Keyaudit (Format 1)

```
SYSID   =   SAMP
AGROUP  =   MAUD
SEGID   =   CD
FIELD   =   CLOR
KANAME  =   ALT
```

- will generate a key of -

SAMPMAUDSACDCLOR

DEFINING AN AUDIT SUBROUTINE

The next two statements are used to define the key of an audit subroutine (format 4). Refer to "Writing Audit Subroutines" on page 3-34 for more information.

SUBNAME = 16-character name of the subroutine key.

This statement is required when generating a subroutine accessed through a 'CALL' statement from a main routine. If this statement is used, the statements shown above are ignored. The SUBNAME operand must be enclosed in quotes.

PARMNAME = One- to four-character name which designates the audited field. This specifies a name which can be used within the following audit statements to refer to the current audited field. Since a subroutine can be called from many routines, which are keyed on different audited fields, this parameter provides a unique name to describe the audited field.

This statement is valid only when SUBNAME has also been specified.

Example:

```
SUBNAME = 'SUBROUTINE NAME '
PARMNAME = AUDF
```

- will generate a key of -

SUBROUTINE NAME

In the audit language statements following these subroutine identifiers, the audited field of the calling routine will be used where AUDF is coded. Refer to "Writing Audit Subroutines" on page 3-34 for more information.

DEFINING A TABLE

The next statement is used to define the 16-character key under which only tables will be specified (format 5). If this form of key is used to specify tables, no Call or Phase identifier may be coded, and only tables may be specified under this key.

TBLROOT = 16-character name of key enclosed in quotes.

Example:

TBLROOT = 'ROOT OF TABLE '

- will generate a key of -

ROOT OF TABLE

The table definition statements and table contents that follow this key identifier will be put into the table portion of the Audit Data Base under the above key.

DEFINING WHEN AUDITS OCCUR

During execution, an auditor module controls all audit operations within the system. This module is called by either a Transaction Driver (Conversational, Nonconversational, or Batch) or a Special Processing Routine. The point at which the Auditor is called will determine which audit logic is executed.

Call Identifier Statements

The Auditor can be called at four points within the transaction. These Call points are:

1. Primary Key Selection - immediately after the key information is entered. This call is used to process keys prior to segment retrieval.

2. Secondary Key Selection - immediately after accessing a segment to be displayed on the Secondary Key Selection screen (Conversational only). This call can be used to prohibit the display of specific segments on the Secondary Key Selection screen or to eliminate access of additional segments.

3. Preaudit - immediately after the DBPATH segment(s) have been loaded in the SPA area. This call is used to validate and process data prior to screen display or batch update.

4. Standard Audit - immediately after the segment data has been modified from the screen or batch input. This call is used to validate and process data prior to applying the updates to the physical record in the data base. If the transaction is Special Processing the Standard Audit call must be made from the Special Processing routine as described in Chapter 6, "Special Processing and User Exits" on page 6-1.

Once the Auditor is invoked, it will determine (through internal controls) the transaction point at which it is called. The Auditor uses this information to determine which portion of the audit routine to execute. For example, a single field can trigger audit logic at Keyaudit, Preaudit, and Standard audit time. Since all logic for a field is maintained under the same keys, a further segregation of each part is needed.

Call identifier statements specify the point at which the Auditor will be called to execute the logic routine. These statements, combined with the following Phase identifiers, cause audit operations to be generated which place each set of logic in the correct position within the Audit Data Base. The four call identifier statements are:

KEY - For audits to be executed at Primary or Secondary Key Selection.

PRELIM - For audits to be executed at Preaudit.
PROCESS - For audits to be executed at Standard audit.
TABLES - For definition of tables to be used by other audits.

At least one Call identifier is required for each generation. However, all of them can be present. They must appear in the sequence shown above.

Phase Identifier Statements

Some audit logic routines are controlled further by another statement called a Phase identifier. These identifier statements further define where the routine resides within the Audit Data Base. There is a close relationship between the Rules Generator operands which specify auditing and the Call and Phase statements. The Rules Generator specifications define when to call the Auditor (Call) and where to find the audits (Phase).

Figure 3-1 shows the relationship between these Rules Generator operands and the Audit Language Call and Phase statements. It may also be helpful to refer to a detailed description of the execution of the three phases given in Chapter 5, "Audit Logic Processing."

RULES GENERATOR OPERAND		CALL	PHASE
KAUDIT = P	1	KEY	P0
KAUDIT = S	2	KEY	P1
PAUDIT = Y	3	PRELIM	P1
AFA = Y	5	PROCESS	P0
AUDIT = Y	4	PROCESS	P1
MSG = Y	6	PROCESS	P2

Figure 3-1. Auditor operands and Audit Language statements

1 Primary Key audit
2 Secondary Key audit
3 Preaudit
4 Standard audit controlled by a field changed
5 Standard audit controlled by the Input transaction rule
6 Standard audit controlled by a field changed

The normal use of more than one phase within a single audit call occurs during Standard audit. Since the P2 phase is only executed when no errors are found during P0 or P1, it may be used to perform final calculations or data base updates. This phase is also used during Standard audit when messages are sent to Project/Groups or userids.

As a general rule Preaudit uses only the P1 phase and the key selection calls are P0 phase for Primary and P1 phase for Secondary.

The Phase Identifier statements are written as follows:

P0 - Defines the P0 phase.
This phase is used for Primary Key selection and Phase 0 of Standard audit
P1 - Defines the P1 phase.
This phase is used for Secondary Key selection, Phase 1 of Standard audit and Preaudit
P2 - Defines the P2 phase.
This phase is used for: Phase 2 of Standard audit

With the exception of the TABLES identifier, at least one Phase identifier must occur under each Call identifier but cannot be repeated. Phase identifiers must occur in the sequence P0, P1, P2. No Phase identifiers may be coded with the TABLES identifier.

Through the use of the definition statements, the user defines audit routines and audit programs within a single generation. An audit routine consists of all audit operations contained under a Call identifier/Phase identifier definition. An audit program consists of all audit routines contained under a specific key as defined by the above statements. The ADFID statement or definition statements for a key can be changed within a generation. If one of these statements is encountered within a generation, the language processor completes processing the previous statements and initiates a new generation based on the new information. Example of defining an audit program:

```
        SYSID   =   SAMP        (NOTE 1)
        AGROUP  =   MAUD        (NOTE 1)
        SEGID   =   CD          (NOTE 1)
        FIELD   =   CLOR        (NOTE 1)

KEY                         Key Audit
 P0                         Primary Key Selection   (NOTE 2)

  Audit Language Code

 P1                         Secondary Key Selection (NOTE 2)

  Audit Language Code

PRELIM                      Preaudit
  P0                        phase P0

   Audit Language Code

  P1                        phase P1

   Audit Language Code

  P2                        phase P2

   Audit Language Code

PROCESS                     Standard Audit
  P0                        phase P0

   Audit Language Code

  P1                        phase P1

   Audit Language Code

  P2                        phase P2

   Audit Languag. Code

TABLES                      TABLE

   Table Definition(s)

FIELD = DQTY                    Identify a new audited field
                                (NOTE 3)

PROCESS                     Standard Audit
  P1                        phase P1

   Audit Language Code
```

Notes:

1. The key for the preaudit and standard audit routines is SAMPMAUDSACDCLOR, and includes table entries under this key.

2. The key for the Key Audit routines is KEYAUDITSACDCLOR

3. The new key (starting with FIELD = DQTY) is SAMPMAUDSACDDQTY

STATEMENTS THAT CONTROL LOGIC FLOW

The logic flow within a series of audit operations is controlled by a True/False branch indicator associated with each operation. If the operation does not contain a decision it will always take the true branch which is the next operation. For example, a move operation will always go to the next operation after completion. A decision operation will take either a true or false branch depending upon the outcome of the operation. The following statements are used to control the branching logic, either conditionally through true or false expressions or unconditionally by a GOTO or CALL statement.

Conditional control statements are divided into two basic statement types, decision and iteration. The 'conditional expressions' referred to in these statements are described later in this chapter.

IF Statements

The decision control statements are defined by IF, ELSE and ENDIF statements. They are written as follows:

```
IF (Conditional Expression)                    Required
    Audit Language Statements
<ELSE                                          Optional
    Audit Language Statements >
ENDIF                                          Required
```

As shown above, the IF and ENDIF are required while an intervening ELSE and its related statements are optional. The presence of the IF, ELSE and ENDIF statements control the next true and next false branches in the generated output. If the conditional expression following the IF is true, the next operation following the IF is executed. Otherwise the next statement following the ELSE (if used) or ENDIF is executed.

IF statements may be nested 50 deep.

Example 1: Two fields are compared and if equal a third field is moved to a fourth. If not equal, the audited field is marked in error and the audit stops.

```
    (SACDCLOR is the audited field)

IF  SACDDEC1 = SACDDEC2
    SACDDEC4 = SACDDEC3
ELSE
    ERRORMSG = 0900
ENDIF
```

Example 2: Two fields are compared and if equal a third field is moved to a fourth. If not equal, a fifth field is moved to the fourth. Processing continues with the statement after the ENDIF.

```
IF  SACDDEC1 = SACDDEC2
    SACDDEC4 = SACDDEC3
ELSE
    SACDDEC4 = SACDDEC5
ENDIF
```

Example 3: Two fields are compared and if equal a third field is moved to a fourth. Then a fifth field is compared to the first and if equal the fifth field is moved to the fourth. If the first IF statement is false, no moves take place. If the second IF statement is false, only the first move takes place.

```
IF  SACDDEC1 = SACDDEC2
    SACDDEC4 = SACDDEC3
 IF  SACDDEC1 = SACDDEC5
     SACDDEC4 = SACDDEC5
 ENDIF
ENDIF
```

DO WHILE Statements

The DO WHILE and ENDDO statements are written as follows:

```
DO WHILE (Conditional Expression)
    Audit Language Statements
ENDDO
```

Within a DO WHILE statement, the conditional expression test is made at the beginning of the iteration. While the conditional expression is true the operations following the DO WHILE are executed. The last statement before the ENDDO statement points back to the DO WHILE statement.

For example:

```
DO WHILE SACDDEC1 < 10
            .
            .Audit Language Statements
            .
    SACDDEC1 = SACDDEC1 + 1
ENDDO
```

The field named SACDDEC1 is compared to 10. If it is less than 10, the operations following are executed and the field is incremented by 1. The statement that increments by 1 also passes control back to the first compare. When the field is equal or greater than 10 the loop is finished and the operation following the ENDDO is executed.

DO WHILE and IF statements can be nested within each other up to 50 deep. For example:

```
IF (Conditional Expression)
    DO WHILE (Conditional Expression)
             (Procedural Statements)
        IF   (Conditional Expression)
             (Procedural Statements)
        ELSE
             (Procedural Statements)
        ENDIF
    ENDDO
ENDIF
```

The ENDIF and ENDDO statements must be supplied. An unequal count of IF to ENDIF statements and/or DO WHILE to ENDDO statements causes a generation error.

An ELSE statement should not be used without a corresponding IF statement. For example:

```
IF (conditional expression)
    DO WHILE (conditional expression)
        (Procedural Statement)
        ELSE
        (Procedural Statement)
    ENDDO
ENDIF
```

The sample sequence is <u>invalid</u> because the ELSE statement falls within the DO WHILE iteration and cannot be associated with an IF statement.

Twin Processing Statements

Another type of iteration control is defined by SETTWIN, SETARRAY, DOTWIN and ENDTWIN statements. These statements combine to provide support for a function called twin processing. A detailed description of the Auditor processing of these operations is provided in "Twin Occurrence Processing" on page 5-39. The statements are written as follows:

```
SETTWIN = 'a list of up to 100 segid's'

SETARRAY  = (field,OFF)

DOTWIN  = numeric constant TO numeric constant
                          or
     related field TO related field

        Audit Language Statements

ENDTWIN
```

The SETTWIN statement defines up to 100 two-character segment IDs in a list format. These segment IDs have been included in the transaction through either the Rules Generator TWIN or TSEG operands. The DOTWIN selects the segment ID whose position in the list is determined by an implicit variable. This implicit variable is initially set to the first numeric constant. Note, both constants can be in related fields. However, if one is in a related field, then the other must be also. The following audit language statements with the selected segment ID is executed until the ENDTWIN is reached. Then, the implicit variable is incremented by one and tested against the second constant. If the implicit variable is greater than the second constant, control is passed to the statement following the ENDTWIN. If the result is less than or equal, control is transferred back to the statement after the DOTWIN.

Example : In this example there are four segments with identical layouts whose IDS are AA,BB,CC,DD. They are defined through the SETTWIN statement. The DOTWIN specifies that the iteration should begin with the first segment (AA) in the list and continue through the fourth segment (DD). All statements between the DOTWIN and ENDTWIN which reference a field contained in each segment (DEC4) have the pointers to that field adjusted on each iteration (i.e. 1st DEC4 in AA, 2nd DEC4 in BB etc.). The result is that field DEC4 in each segment is replaced with SACDDEC6 if it originally is equal to SACDDEC5.

```
SETTWIN = 'AA,BB,CC,DD'
DOTWIN = 1 to 4
 IF  SAAADEC4 = SACDDEC5
     SAAADEC4 = SACDDEC6
 ENDIF
ENDTWIN
```

If the DOTWIN statement is written 'DOTWIN = 3 to 4' only segments CC and DD will be changed.

The SETARRAY statement defines a field which is associated with the first segment ID defined in the SETTWIN statement. It is assumed that the field name specified is the first of a set of identical fields in a single segment. These fields are associated one for one with each segment ID specified in the SETTWIN statement. The OFF parameter causes the association of the array of fields and twin segments to be broken.

Example : In this example the same four segments shown above are used. The SETARRAY references a field SACXDEC1 which is assumed to be the starting field of a single level array containing at least four fields. Each iteration through the DOTWIN/ENDTWIN loop causes the array pointer to be shifted in the same manner as the segment pointer. The result is that field DEC4 in each segment is replaced with a corresponding array field if it originally is equal to SACDDEC5.

```
 SETTWIN = 'AA,BB,CC,DD'
SETARRAY =  SACXDEC1
 DOTWIN = 1 to 4
```

```
 IF  SAAADEC4 = SACDDEC5
     SAAADEC4 = SACXDDEC1
 ENDIF
ENDTWIN
```

GOTO Statement

The GOTO statement is coded as follows:

```
GOTO    label name
```

The label name is a 1- to 8-character alphanumeric string with no embedded blanks. The statement on which the label is attached is written:

Label name: Audit Language Statement

Note: The label must be immediately followed by a colon.

Example:

```
(SACDCLOR is the audited field)

GOTO COMPUTE1
     .
     .Audit Language Statements
     .

COMPUTE1: CLOR = SACDDEC1
```

The GOTO statement generates a NOP operation which branches directly to the appropriate sequence.

CALL Subroutine Statement

Control can be passed from one audit routine to a subroutine through a CALL statement. The called routine takes on the attributes of the calling routine. This means that a reference to the audited field in a subroutine will point to the audited field of the calling routine. Subroutine call can be nested 5 deep.

The CALL statement is coded as follows:

```
CALL <=> 'subroutine name <,XX>'
```

The subroutine name is the 16-character name of the key of the subroutine (see SUBNAME under "Defining an Audit Subroutine" on page 3-5). The user can optionally branch to a specific sequence key within the subroutine by specifying the two character key 'xx'. Otherwise, the branch is made to the first operation in the subroutine. The entire parameter must be enclosed in single quotes. Generation of a subroutine is discussed in "Writing Audit Subroutines" on page 3-34

For example:

```
CALL    'SUBROUTINE   NO1'
```

Control is transferred to the first operation in the subroutine.

Example:

```
CALL    'SUBROUTINE   NO1,15'
```

Control is transferred to the operation with a sequence key of 15 in the subroutine 'SUBROUTINE NO1'. A specific sequence key within a subroutine is determined by using the compiled listing for that subroutine.

The subroutine must reside in the phase corresponding to that of the calling routine. This means for example, a subroutine called from a

Primary Key Selection audit must be defined in Phase P0. Control is normally returned to the next operation descriptor following the one executing the call. Therefore, an operation descriptor should be present to receive control back from the subroutine.

EXIT Statement

The EXIT statement is coded as follows:

```
EXIT
```

This statement successfully terminates the current audit routine.

STOP Statement

The STOP statement is coded as follows:

```
STOP
```

This statement terminates the current call to the auditor. No subsequent field audits will be executed during the current auditor call.

CONDITIONAL EXPRESSIONS WHICH COMPARE AND TEST

Conditional expressions provide the capability to vary logic within the audit routines. These expressions are always preceded by the statements IF or DO WHILE and control the true and false branching of the logic.

In some cases the conditional expressions perform an operation other than a compare and branch. This type of conditional expression is explained in "Conditional Expressions which also perform an operation" on page 3-18. This section explains the type of conditional expressions which generate only a compare or test operation. They are:

- compare one item to another.
- compare one item to a list or range.
- compare for type of data content.
- compare against system values.
- test switch settings.

COMPARING ONE ITEM WITH ANOTHER

Compare expressions are coded in the following format:

```
ITEM1   (Relational Operator)   ITEM2
```

ITEM1 or ITEM2 can be:

- an audited or related field,
- accumulation or auxiliary counter, or
- literal or numeric constant.

The valid relational operators are:

=	or	EQ	Equal
¬=	or	NE	Not equal
<	or	LT	Less than
>	or	GT	Greater than
<=	or	LE	Less than or equal (negative of GT)
>=	or	GE	Greater than or equal (negative of LT)

If a compare expression is true, the operation following the compare is executed. If it is not true, the false branch is taken. The false branch for a compare expression preceded by an IF is the operation specified immediately following the ELSE or ENDIF statement associated

with it. The false branch for a compare expression preceded by a DO WHILE is the operation following the associated ENDDO statement.

Example: This example first checks field SACDDEC2 greater than 10. If it is, the audited field CLOR is moved to another field SACDDEC1. If it is not greater than 10, the audited field is iteratively added to SACDDEC1 until the value in SACDDEC2 reaches 10.

```
(SACDCLOR is the audited field)

IF  SACDDEC2 GT 10
    SACDDEC1 = CLOR
ELSE
    DO WHILE SACDDEC2 < 10
        SACDDEC1 = SACDDEC1 + CLOR
        SACDDEC2 = SACDDEC2 + 1
    ENDDO
ENDIF
  .
  .
  .
```

COMPARING A FIELD TO A LIST OR A RANGE OF VALUES

Multivalue compare expressions are coded in the following format:

```
field   <NOT> IN    (List, Range)
```

Field refers to an audited or related field. A list is specified as a series of literals or numeric constants separated by commas. The 'IN' operand directs flow to the next statement if the field value is in the list or range(s) specified. The 'NOT IN' operand directs flow to the next statement if the field value is <u>not</u> in the list or range(s) specified.

```
Literals            -   'AB,CD,EF'    or    'AB','CD','EF','  ','XY'
Numeric constants   -   11,12,13,14,15
```

A range is specified as a series of upper/lower values separated by commas. Each set of upper/lower values is separated by a colon.

```
Literals            -   'AB':'CD','GH':'KL'
Numeric constants   -   100:200,400:500
```

When performing a range check, the IN <NOT IN> operator must be used. A list check can use either the IN <NOT IN> operators or the EQ/NE relation described above.

Example: In this example the audited field is checked against two ranges. If it falls within either one, its value is set to 1000. Next a field SACDDEC2 is checked for 3 literal values. If it does not contain one of the values, it is set to 'DD'.

```
(SACDCLOR is the audited field)

IF CLOR IN  100:200,500:600
   CLOR = 1000
ENDIF

IF SACDDEC2 NOT IN 'AB,AC,AD'
   SACDDEC2 = 'DD'
ENDIF
```

TESTING A FIELD FOR DATA TYPE

Data type expressions are coded in the following format.

```
field   (=,¬=,EQ,NE) Type of Content <literal>
```

Field refers to an audited or related field. The valid types of data are:

Type	Field Contains
ALPHA	Alphabetic (A-Z) no blanks
NUM	Numeric (0-9) no blanks
ANUM	Alphanumeric - no blanks
ALPHAB	Alphabetic (A-Z) and blank
NUMB	Numeric (0-9) and blank
ANUMB	Alphanumeric and blank
EMBEDB	Embedded blanks within the field
NOEMBEDB	No embedded blanks within the field
DBCS	Valid DBCS character(s)
MASK	Check against list mask
NON$	Check against $$$ mask

The term embedded blanks does not mean that the field contains no blanks. It means that a blank in a field is followed by a character. For example:

```
'AA  BB'  has two embedded blanks
' BBBBB'  has one embedded blanks
'AAA   '  has no embedded blanks
```

MASK and NON$ require a literal in the following formats:

```
MASK    -   'A,B,C,D'   or  '1,2,3,4'
```

MASK checks each one-character literal in the mask against each byte within the field. If any literal in the mask is found in the field, the result is equal. If a field contains AB123 and the MASK = 'D,E,F,A' the result is true; if MASK = 'D,E,F,4' the result is false.

```
NON$ - '$$56$,$$57$'
```

NON$ uses the mask to check the non-dollar sign characters against the same relative position within the field. If <u>all</u> non-dollar sign characters match the outcome is equal. If a field contains AB123 and the NON$ mask = '$$1$$' the result is true; if NON$ = '$1$$$' the result is false. Multiple NON$ masks, separated by commas, can be specified.

Example: This example first checks the field SACDDEC2 for all numeric data. If it is numeric then the field is checked for either a 12 or 13 in positions three and four of the field. If either value exists, the audited field is set to 10.

```
(SACDCLOR is the audited field)

IF  SACDDEC2 = NUM
    IF  SACDDEC2 EQ NON$ '$$12$,$$13$'
        CLOR = 10
    ENDIF
ENDIF
```

TESTING THE VALUE OF A SYSTEM FIELD

System field are maintained by IMSADF II in the Scratch Pad Area (SPA) during the execution of a transaction. These fields either define information about the transaction, or contain a status code from a previous DL/I or DB2 call. System field expressions are coded in one of the following three formats:

1. system field (=,¬=,EQ,NE,<NOT> IN) list

   The system field keywords to use with Format 1 are:

Keywords	Description
PGROUP	two-character Project Group code
USERID	six-character user Sign-on ID
MODE	one-character transaction mode (1-6)

TRXID	two-character transaction ID
LTERM	eight-character logical terminal name
STATCODE	two-character DL/I Status Code (from last DL/I call)
SYSID	four-character system ID
SPAULANG	one-character national language code
RETRIEVAL	list of retrieved segment IDs

The first eight system fields above cause the current contents of that field to be compared against a single literal or numeric constant or against a list of literals or numeric constants. If a match is found the result is equal. The RETRIEVAL keyword causes a single segment ID or a list of segment IDs to be checked against all currently loaded segments. If any segment in the list has been retrieved the result is equal.

Example: If an occurrence of either segment ID A1 or A2 has been retrieved and the transaction is in the add mode, the audited field is set to 10.

```
(SACDCLOR is the audited field)

IF  RETRIEVAL = 'A1,A2'
    IF MODE EQ 4
        CLOR = 10
    ENDIF
ENDIF
```

Example: By changing the relational operator to ¬= the same results occur when either A1 or A2 have not been retrieved.

```
(SACDCLOR is the audited field)

IF  RETRIEVAL ¬= 'A1,A2'
    IF MODE EQ 4
        CLOR = 10
    ENDIF
ENDIF
```

2. SQLWARNn (=,¬=,EQ,NE) (' ','W')

   where n is 0 to 7

This system field contains eight warning indicators which are passed back by DB2 after a SQL call. Refer to SQLWARN in the <u>IMS Application Development Facility II Version 2 Release 2 DATABASE 2 Application Specification Guide</u> for details about this field. Use of one of the SQLWARNn keywords causes the corresponding system field to be checked for a value of ' ' or 'W'.

3. SQLCODE (=,¬=,>,<,>=,<=,EQ,NE,GT,LT,GE,LE) list

   where list is one or more numeric constants separated by commas

SQLCODE contains the status of the last DB2 SQL call. This is comparable to the DL/I status code maintained in STATCODE. Use of SQLCODE causes a compare between the corresponding system field and a list of one or more numeric constants. If any one compare is true, the result of the compare is true. It is recommended that only one constant be coded for a compare using GE, GT, LE, and LT.

Example: If the SQL call did not complete correctly, stop the audit with an error 0001. If a not found condition occurs stop the audit with an error 0100. If the SQL call completes successfully but a warning is returned, stop the audit with an error 1000.

```
IF SQLCODE < 0
  ERRORMSG = 0001
ELSE
  IF SQLCODE = 100
    ERRORMSG = 0100
  ELSE
```

```
      IF SQLWARN0 = 'W'
        WARNMSG = 1000
      ENDIF
    ENDIF
  ENDIF
```

TESTING FIELD INDICATORS

Indicators exist which determine the status of a field or a switch. These indicators can be tested and logic varied by the results. Field indicator expressions are coded in the following format:

```
<field> indicator (=,¬=,EQ,NE) (ON,OFF)
```

The types of indicators which can be tested are:

BIT	-	If a field type is BIT a test is made for 0 (OFF) or 1 (ON).
NULL	-	If the field's indicator variable is <0(ON) or 0(OFF). Valid for DB2 fields only.
TRUNC	-	If the field's indicator variable is >0(ON) or 0(OFF). Valid for DB2 fields only.
CHANGED	-	If the field has been modified (ON) or not modified (OFF).
ERROR	-	If the field has been previously marked in error (ON) or not (OFF).
SWITCHi	-	Test the internal SWITCHi for ON or OFF status, where i = 1 through 9. These internal switch are maintained by the Auditor and can be set and tested through the audit language. They are initialized upon entry to the Auditor.

When testing BIT, CHANGED, ERROR, NULL or TRUNC indicators, the field operand must be specified for a related field. If not specified, the field name defaults to the audited field.

Example: If the bit field SACDBITT is ON, stop with error 7777. If field SACDDEC2 has not been marked in error, stop with error 8888. If switch # 5 is on stop with error 9999.

```
(SACDCLOR is the audited field)

IF  SACDBITT BIT = ON
    ERRORMSG = 7777
ENDIF
IF  SACDDEC2 ERROR = OFF
    ERRORMSG = 8888
ENDIF
IF SWITCH5 EQ ON
    ERRORMSG = 9999
ENDIF
```

TESTING SEGMENT INDICATORS

An indicator is available to test whether or not a segment has been modified. It is tested by using the following statement:

```
segid CHGEFLAG (=,¬=,EQ,NE)  (ON,OFF)
```

If CHGEFLAG is ON, then the segment has been modified.

USING THE ANYERR STATEMENT

A test can be made to see if there has been any previous error during the current call to the Auditor. The format for the statement is:

```
IF ANYERR (=,¬=,EQ,NE) (ON,OFF)
```

This operation does not check for errors in a previous call to the Auditor, only for errors during the current call to the Auditor.

CONDITIONAL EXPRESSIONS WHICH ALSO PERFORM AN OPERATION

Conditional expressions also can perform an operation and test the results of that operation. These expressions are always preceded by an IF statement and control the true and false branching of the logic. Within these conditional expressions, the following expression types can occur:

- Perform data base calls
- Table translations and Searches
- Call user audit exits

DL/I CALLS

DL/I calls may be issued against segments defined either in DBPATH or TSEGS. These data base accesses are executed immediately or at transaction termination depending upon the operands specified.

DL/I call expressions are coded in the following format:

```
inst <IMMED> <field,KEYFIELD> <'>segid<'> <NOT> OK
```

inst defines the type of DL/I call required. The valid types are:

DLET	GHU	GNPQ	GU1	ISRT
GHN	GHUU	GNQ	HDEL	REPL
GHNP	GN	GU	HNPQ	SGN
GHNQ	GNP	GUU	HREP	

These call types are the same ones used in SEGHNDLR calls which are explained in Chapter 6, "Special Processing and User Exits."

IMMED is required if a data base update call is issued for immediate execution. If IMMED is not specified, all updates are performed at the end of the transaction by the transaction driver. IMMED is not needed for retrieve calls. Refer to "DL/I Data Base Operations" on page 5-33 in Chapter 5, "Audit Logic Processing" for additional information on the use of immediate data base operations.

field specifies the name of the field which contains the concatenated key of the target segment associated with the DL/I call.

Note: For the function HREP or HDEL, if the target segment associated with the DL/I call has been retrieved, then the key associated with that segment will be used in the call. Otherwise, the key passed in the KEYFIELD will be used. To override a retrieved segment; turn off the retrieve flag before issuing the DL/I call and then the key passed in the KEYFIELD will be used.

If the current key of a retrieved segment is required the keyword KEYFIELD should be used. Otherwise, 'field' should specify the name of an audited or related field.

segid specifies the two-character segment ID of the target segment of the DL/I call.

The DL/I call expression performs both a DL/I call and a check of the DL/I status code returned from the call. If the DL/I status code is blank, the NEXT TRUE branch is taken; otherwise, the NEXT FALSE branch is used. If 'OK' is specified, the NEXT TRUE branch points to the next statement. If 'NOT OK' is specified, the NEXT FALSE branch points to the next statement. To determine the specific non-blank status code returned a system expression with the STATCODE keyword can be used.

Example: If the status code of a Get Unique call to the 'AA' segment is blank, SACDDEC1 is set to 1. If not blank, and the segment could not be found, SACDDEC1 equals 2. Finally, if not blank and any other bad status, SACDDEC1 equals 3.

```
(SACDCLOR is the audited field)

IF GU SACDKEY1 AA OK
    SACDDEC1 = 1
ELSE
    IF STATCODE = 'GE'
        IF ISRT IMMED SACDKEY1 AA OK
            SACDDEC1 = 2
        ELSE
            SACDDEC1 = 3
        ENDIF
    ENDIF
ENDIF
```

VSAM CALLS

Calls may be issued against VSAM files defined either in DBPATH or TSEGS. A call against a VSAM file is coded the same as a DL/I call.

For a KSDS or RRDS data set, the following call types are valid:

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

For an ESDS data set, ISRT is the only valid type.

SQL CALLS

SQL calls may be issued against tables defined either in DBPATH or TSEGS. These data base accesses are executed immediately. These calls are the same ones used in SQLHNDLR calls which are explained in Chapter 6, "Special Processing and User Exits."

SQL call expressions are coded in the following format:

```
SQL label field <'>tableid<'> <NOT> OK
```

SQL identifies the statement as a SQL call function

label defines the type of SQL function, either standard or user supplied

Following are the valid SQL functions that may be specified. All operations are immediate.

SELECT, UPDATE, DELETE, INSERT
CSELECT, CUPDATE, CDELETE
KSELECn0, KSELECnF, KSELECnC
CSELECTO, CSELECTC
CUPDATEO, CUPDATEU, CUPDATEC
CDELETEO, CDELETED, CDELETEC

Refer to <u>IMS Application Development Facility II Version 2 Release 2 DATABASE 2 Application Specification Guide</u> for a detailed explanation of these SQL functions.

USER SUPPLIED - where label is 1 to 8 characters and is the label specified as the SQLCALL in the generate statement when OPTIONS=TABH and SQLUSER=YES.

If the table operation should not be immediate (i.e., should occur after successful PROCESS Audits), the SETFLAG operations should be used. Also note that data compare (DATACOMP) only applies to transaction driver updates or a SEGUPDTE exit function call.

field name of the field containing the search values. 'field' must represent a field containing the concatenation of all host variables in the WHERE clause. For the standard SQL functions, the host variables are in the order in which the KEY=YES columns are specified to the Rules Generator. A keyword 'KEYFIELD' specifies that the key associated with the current retrieved row is used. For the user supplied SQL functions, the 'field' must must represent a concatenation of the host variables in the WHERE clause and in the order specified in the WHERE clause.

tableid specifies the two-character table ID of the DB2 table to be accessed.

OK OK indicates that SQLCODE was 0 and SQLWARN0 was not 'W'.

Example: Retrieve a row from table ES using a key contained in field SACDKEY. If successful, add 1 to SAESCNT1. Then update the occurrence using the current key. If there is an error during update, stop the audit with an error code of 0805.

```
IF SQL CSELECT SACDKEY ES OK
    SAESCNT1 = SAESCNT1 + 1
  IF SQL CUPDATE KEYFIELD ES NOT OK
    ERRORMSG = 0805
  ENDIF
ENDIF
```

USING ENCODE/DECODE TABLES

Tables of arguments and values can be created and accessed through the audit language. Refer to "Creating Tables" on page 3-32 for information on creating these tables. Once created, the tables can be used to perform Encode and Decode operations or be used in a table lookup operation. Tables operations are coded in the following formats:

1. (ENCODED,DECODED) field1 TO field2 USING ('tablename',relfield) <NOT> OK
2. field <NOT> IN (ARG,VAL) ('tablename',relfield)

The first format encodes or decodes a value from an audited field to a related field or vice versa. One field must be the audited field while the other field must be a related field.

Field1 names a field containing a value which is used to search the table argument (if DECODED) or function (if ENCODED).

Field2 is replaced from the table with the argument value (if ENCODED) or function value (if DECODED) from the table. A search field or a receiving field using an argument value can be 1-8 characters in length with a data type of alphabetic, alphanumeric, numeric, or decimal. A search field or a receiving field using a function value can be 1-70 characters in length with a data type of alphabetic, alphanumeric, numeric, or decimal. Refer to 'Table Identification and Specification' in this chapter for a discussion on building the tables.

Tablename is a 22-character table key which includes a 16-character root key and a six-character table in ID key. If blanks occur in the root or table ID key they must be shown in the tablename. The tablename can be a 22-character literal or a related field which contains a 22-character literal. Once the encode or decode is accomplished a true or false branch is taken depending upon successful or unsuccessful completion. If the search field (Field1) is not found the encode/decode operation is not performed and a false branch is taken.

The second format is used to search the table either for an argument (ARG) or function (VAL). If found, a true branch is taken and if not

found a false branch is taken. Encode or decode is not performed on this format.

Example: The value in the audited field is used as an argument to search the table. If a match is found, the corresponding value is moved to the field SACDDEC1. Then the value in SACDDEC2 is used to search the same table and if not found the audit is stopped with error 9999.

```
(SACDCLOR is the audited field)

IF  ENCODED CLOR TO SACDDEC1 USING
    'TABLECODES      TABLE1' OK
    IF  SACDDEC2 NOT IN ARG
        'TABLECODES      TABLE1'
        ERRORMSG = 9999
    ENDIF
ENDIF
```

CALLING A USER EXIT

User written audit exits can be accessed through the Audit Language. The creation of these exit modules is explained in "User Audit Exit Processing" on page 6-52. Control is returned from the exit to the operation making the call. A code, returned from the exit logic, determines whether the true or false branch is taken.

User exit expressions are coded in the following format:

```
AEXIT <'>code<'> <related field> <PASS(literal,numeric constant)
    RETURN (=,¬=,EQ,NE) (TRUE,FALSE)
```

Code specifies the user exit code which ranges from 70-99 or W0-Z9. The Auditor will always make an audit exit call when it detects an operation with a code in these ranges. Different codes can represent different functions in the audit exit module. The exit module should be written to branch to the appropriate routine depending upon the passed code. The name of a related field can be passed to the exit routine on the call by specifying a five- to eight-character (RF) name which may be enclosed in parentheses. This field may contain variable information created during the audit.

The PASS parameter specifies literal data values to be passed to the audit exit. Note that the () are not coded but indicate a literal, enclosed in quotes, or a numeric constant must follow.

Upon return from the exit, a true or false branch is taken depending upon the status of a true/false indicator. This indicator is set by the exit prior to return. The RETURN operands specifies whether the true or false return is used to control branching.

Example: A call is made to the audit exit and code 75 is passed. Upon return the true/false indicator is checked. If set to true another exit call is made. This time code W5 is passed and a pointer to field SACDDEC2 is passed. If the true/false indicator is set to false the audit is stopped with error code 7777.

```
(SACDCLOR is the audited field)

IF  AEXIT 75 RETURN = TRUE
    IF AEXIT W5 SACDDEC2 RETURN EQ FALSE
        ERRORMSG = 7777
    ENDIF
ENDIF
```

SETTING MESSAGES

Messages are triggered through the Audit Language. These messages take the form of errors (which stops the audit on a field), warning (which give the operator the option of continuing) or informational (which are sent to a mailbox or the screen).

Messages are broken into two categories;

1) Those that stop the audit
2) Those that let the audit continue

MESSAGES THAT STOP THE AUDIT

In the first category, there are three statements provided to set error and informational messages and stop the audit.

Audit errors are flagged during the P0 and P1 phases of all audit calls. Audit errors cause the current audited field to be marked in error, and the associated error message number to be appended to it. Auditing stops on that field and continues on the next field.

Informational messages are flagged only in the P2 phases of Preaudit and Standard audit calls. The audited field is marked for message sending and the associated message routing number is appended to it. Auditing stops on that field and continues on the next field.

These message are set as follows:

```
ERRORMSG    <=>    message number
INFOMSG     <=>    routing number
MSG         <=>    message number/routing number
```

ERRORMSG is a valid statement under the phase identifiers of P0 and P1. Specifying ERRORMSG causes the associated one- to four-digit message number to be associated with the field definition. It also causes the field to be flagged in error and stops the audit. If ERRORMSG is used under phase identifier P2, a listing error is generated.

INFOMSG is a valid expression under phase identifier P2. This expression causes the associated four-character routing number to be placed with the field definition. It also causes the field to be flagged for message sending and stops the audit. If INFOMSG is used under phase identifier P0 or P1 a listing error is generated.

MSG can be used in place of ERRORMSG and INFOMSG throughout the program. Depending upon the phase in which MSG is encountered, the field is flagged in error or flagged for message sending.

These expressions are used in conjunction with conditional expressions as follows:

```
IF  (Conditional Expression)
            .
            .Audit Language Statements
            .
ELSE
    ERRORMSG = 9999
ENDIF


            or

IF  (Conditional Expression)
    ERRORMSG = 9999
ENDIF

        or

IF (Conditional Expression)
    ERRORMSG = 9999
ELSE
    •
```

```
        • Audit Language Statements
        •
    ENDIF
```

Depending upon the conditional expression and the placement of the ERRORMSG/INFOMSG or MSG expression, either the true or false branch is set to stop the audit with an error.

Example: If SACDDEC1 equals 10, stop with error 1111. Otherwise, if SACDDEC1 = 20, set it to 30. If not 20, stop with error 2222.

```
    IF  SACDDEC1 = 10
        ERRORMSG = 1111
    ELSE
        IF  SACDDEC1 = 20
            SACDDEC1 = 30
        ELSE
            ERRORMSG = 2222
        ENDIF
    ENDIF
```

MESSAGES THAT DO NOT STOP THE AUDIT

Other messages can be set while allowing the processing logic to continue. These statements allow the messages to be appended to a related field. This allows multiple error or warning messages to be generated from a single audit routine.

Setting a Warning Message

Warning messages are used to display information prior to final data base update. The terminal user then has the option of continuing with the update or suspending it.

```
    WARNMSG <relfield> <=> message number
```

This statement specifies that a Warning Message number should be appended to the related field named in relfield. If relfield is not coded, the audited field is used. Processing continues after the message is set. Warning messages are described in Chapter 5, "Audit Logic Processing."

Example: If the audited field equals SACDDEC1, set warning message #0010 on the audited field and continue. If the audited field equals SACDDEC2, override the warning message number with 0020 and continue.

```
    IF  CLOR = SACDDEC1
        WARNMSG = 0010
    ENDIF
    IF  CLOR = SACDDEC2
        WARNMSG = 0020
    ENDIF
```

Setting an Error Message for a Related Field

Error messages can also be set on related or audited fields while continuing the process logic.

```
    SETERRMSG <relfield> <=> message number
```

This statement specifies that a Error Message number should be appended to the related field named in relfield. If relfield is not coded, the audited field is used. Processing continues after the message is set. Error messages are described in Chapter 5, "Audit Logic Processing."

Setting an Informational Message

Informational messages can also be shown on the display screen. This type of message does not flag a field in error or for a warning as the preceding examples do. The message line on the segment display screen is used to display the text of the message.

```
SPAERMSG = 'message text'
```

The message text specified is used to override the normal end of transaction message. Message text can be 1 to 50 characters in length and must be enclosed in single quotes.

NOP STATEMENT

A NOP statement generates a no-operation descriptor which always takes a true branch. It is coded as follows:

```
NOP
```

If a NOP statement is followed by an ERRORMSG or INFOMSG statement the message or routing number is appended to the no-operation as follows:

```
    NOP
ERRORMSG = 9999
```

This causes an unconditional error wherever the operation is accessed.

ASSIGNMENT STATEMENTS

Assignment statements control the movement of data between the audited field, related fields, the accumulation counter, auxiliary counters, system fields, numeric constants and literals. These statements are used to move single fields, parts of fields or full segments between data areas within the product.

BASIC FIELD MOVES

The basic move statement causes the data in one field to be moved to another field. The format of this type of move is:

```
item1 = item2
```

item1 is the target of the move and can be an audited field, related field, accumulation counter, auxiliary counter or some system fields.

item2 is the source of the move and can be an audited field, related field, accumulation counter, auxiliary counter, some system fields or a literal.

A move to the accumulation or auxiliary counters will generate an add of the source field + 0.

Following is an example of statements to move data between various fields and counters.

```
   ACCUM = CLOR
COUNTER1 = CLOR
COUNTER1 = ACCUM
   ACCUM = 100.5
    CLOR = SACDDEC2
```

When data is moved between fields with different data types (i.e., NUM to PD) or field lengths, the data is automatically converted and adjusted for length.

When assigning alphanumeric characters to a hex field, the characters are converted two at a time from the source field to create one hexadecimal byte in the target field.

Refer to Appendix B, "Data Conversions and Mappings" for a complete description of all conversion possibilities.

USING THE CONCAT STATEMENT

The CONCAT statement concatenates two source fields into one target field. The entirety of each source field is used in the concatenation, determined by the field length. Only fields types of ALPHA, NUM, ALPHANUM, and VARCHAR can be concatenated. The fields used can only be audited or related fields.

In the case of a VARCHAR field (DB2 only) the current field length for the field is used. Also for DB2 columns which can be null; if one of the source fields is null, the target field is set to null.

Example: The value in field SAPSSTR2 is concatenated behind the value in field SAPSSTR1 and the result is placed in the audited field CLOR.

```
CLOR = CONCAT SAPSSTR1 SAPSSTR2
```

USING THE SUBSTR STATEMENT

The SUBSTR statement allows for movement of portions of fields to and from another field.

There are two forms of the SUBSTR operation. The first form substrings a portion of the source field and moves it into a target field. This format looks like:

```
tgtfield = SUBSTR  srcfield 'starting position' : length
```

The second form moves the entire source field into a substring portion of the target field. This format looks like:

```
SUBSTR tgtfield starting position : length = srcfield
```

Any field name used may be the audited or a related field. Both starting position and length must be numeric constants or both must be fields containing numeric constants. The numeric constants must be 1 to 255.

Example: The data in field BAPSSTR2, beginning at position 3 and continuing for a length of 7, is moved to the audited field CLOR. The numbers could also be contained in fields BAPSSTR3 and BAPSSTR4.

```
CLOR = SUBSTR BAPSSTR2 3 : 7

              or

CLOR = SUBSTR BAPSSTR2 BAPSSTR3 : BAPSSTR4
```

Example: The data in the audited field field CLOR is moved to field BAPSSTR2, beginning at position 3 and continuing for a length of 7. The numbers could also be contained in fields BAPSSTR3 and BAPSSTR4.

```
SUBSTR BAPSSTR2 3 : 7 = STR1

              or

SUBSTR BAPSSTR2 BAPSSTR3 : BAPSSTR4 = STR1
```

Moving Data to or from a System Field

There are specific system fields which can be moved or modified through the Audit Language. If a system field is the source of a move, its current value is moved to the specified target field. The values for Date or Time are obtained from the MVS operating system.

The following system fields can be the source of a move operation:

FIELD	DESCRIPTION
MODE	one-character transaction mode (1-6)
TRXID	two-character transaction ID
PGROUP	two-character project group
USERID	six-character user ID
$TIME	eight-character current time
$DATE	six-character current date
LTERM	eight-character logical terminal name
SPAKEYID	255-character concatenated key of target segment(s)
SYSID	four-character system ID

Example: Move the current time into field SACDDEC3.

```
SACDDEC3 = $TIME
```

The following system fields can be the target of a move operation:

FIELD	DESCRIPTION
MODE	one-character transaction mode (1-6)
TRXID	two-character transaction ID
SPAKEYID	255-character concatenated key of target segment
SPASQL	KSELECTn, where n is 3 to 9
SPAWHERE	field that contains the key selection WHERE host variables
PCBNUM	number from 1-120

Example: Changes the current transaction ID to AB. This causes the current transaction to complete, and the new transaction to be executed.

```
TRXID = 'AB'
```

When some system fields are modified, additional action is taken by the system. For example; changing MODE, TRXID or SPAKEYID causes a transaction switch to occur. Refer to "Transaction Switch Operations" for additional information on modification of MODE, TRXID and SPAKEYID.

Following is information on syntax of SPASQL, SPAWHERE, and PCBNUM statements.

SPASQL represents a system field in the SPA (SPASQLKS) to name the user key selection function to be executed. SPAWHERE represents a system field in the SPA (SPAWHERE) to name the field holding the key selection WHERE host variables. SPAWHERE should be set in conjunction with the SPASQL key selection function if the WHERE clause requires other than the table KEY=YES column values.

Syntax for SPASQL statement:

```
SPASQL = (field,KSELECTn,n)
```

where field is a related or audited field that contains an eight-character name of the format KSELECTn, and n is a digit 3 to 9

If n or KSELECTn is used, the value of n will be stored in the data descriptor for the operation. If a field is used, its name will be in the related field area of the operation descriptor, and there will be no data descriptor.

Example of 3 forms of SPASQL statements:

```
SPASQL = RELFIELD
SPASQL = KSELECT3
SPASQL = 3
```

Syntax for SPAWHERE statement:

```
SPAWHERE = field (where field is audited or related field)
```

Example: Audited field contains the host variables. Related field contains the host variables. The host variables are in the current key area referenced by the keyword 'KEYFIELD'.

```
SPAWHERE = AUDF
SPAWHERE = RELFIELD
SPAWHERE = KEYFIELD
```

Syntax for PCBNUM statement:

Example:

```
<'>segid<'> PCBNUM = (number,related field)
```

This statement dynamically changes the PCB used in retrieving the specified segment ID.

Example: The user data base PCB number is changed to 6 for segment ID CD. This changes the PCB# which is originally set through the Rules Generator operand PCBNO.

```
CD PCBNUM = 6

     or

CD PCBNUM = SACDDEC1
```

INITIALIZING SEGMENTS

Segment areas in the SPA can be set to initialized values through a single initialization statement. Each field within the specified segment ID(s) is initialized to blanks or zeros depending upon its field type. All segment areas in a transaction can be initialized in one statement.

```
INITSEGS = 'segid1,segid2'...
```

USING THE MAP STATEMENT

The MAP statement may be used to move a group of fields that may be in different segments to or from a contiguous area in another segment. The three formats of the MAP statement are:

```
MAP id TO relfield

       OR

MAP relfield TO id

       OR

MAP id TO id
```

Relfield is the name of a related field that is the beginning of a contiguous area in a segment. If a related field is specified, the id that is specified must be a mapping segment id.

If the map is from one id to another, one id must be a mapping segment and the other must be a segment id in the transaction that is not a mapping segment.

The MAP statement is executed by the Auditor with a call to MAPPER in MFC1ESPI. Information about how a MAPPER call works may be found in Chapter 6, "Special Processing and User Exits." Information about how to specify a mapping segment can be found in Chapter 2, "Rules Generator."

USING THE COPYSEG STATEMENT

The COPYSEG statement may be used to move all or part of a segment to or from another segment in the SPA. If part of a segment is used, it must be a contiguous part of the segment. The three formats for the COPYSEG statement are:

```
COPYSEG id TO relfield

       OR

COPYSEG relfield TO id

       OR

COPYSEG id TO id
```

Id is a segment id. Relfield is the name of a related field that is the beginning of a contiguous area in a segment to be used by COPYSEG.

The COPYSEG statement is executed by the Auditor with a call to COPYSEG in MFC1ESPI. Information about how a COPYSEG call works may be found in Chapter 6, "Special Processing and User Exits."

ARITHMETIC STATEMENTS

Arithmetic operations are performed against fields (with an attribute of NUM, BIN, DEC, or PD). All field types are converted to a PD format before the arithmetic operation is performed. All arithmetic is performed in an internal accumulation counter. Automatic decimal alignment is performed as shown in the following example.

	FIELD A	FIELD B	FIELD C	FIELD D	ACCUMULATION CTR
1)	1.1	2.22	3.333	7.7777777	000000000000000
2)					1.1
3)					4.433
4)					2.213
5)					4.91286
6)					1.47400
7)	1.4	1.47	1.474	1.4740000	1.47400

1) Shows the original values in each field or counter.
2) Add field A to accumulation counter.
3) Add field C to accumulation counter.
4) Subtract field B from accumulation counter.
5) Multiply field B by accumulation counter.
6) Divide field C into accumulation counter.
7) Shows moves from accumulation counter to each field.

Arithmetic operations are performed between:

audited fields (AF)
related fields (RF)
accumulator counter (ACCUM)
auxiliary counters (CTR)
numeric constants (DD)

They are coded in the following format:

(AF,RF,ACCUM,CTR) = (AF,RF,ACCUM,CTR,DD) (+,-,*,/) (AF,RF,ACCUM,CTR,DD)

The nine auxiliary counters are specified as COUNTER1...COUNTER9. They are reset to zero upon entry to the Auditor. Values which are derived and stored in one of these counters in one audit routine, are maintained and can be checked in another routine. The accumulation counter is specified as ACCUM. This counter is reset to zero at the beginning of each audit. Numeric constants are numeric digits with an optional decimal point and/or leading minus sign (i.e., 123.45, -5.77).

Example: The field SACDDEC2 is multiplied by 1.5 and the product is placed in the accumulation counter. Then the audited field is added to SACDDEC2 and the sum is placed in the audited field. Finally the value in the accumulation counter is added to auxiliary counter 1 and the sum is placed back in auxiliary counter 1.

```
SYSID = SAMP                    SYSTEM ID
AGROUP = MAUD                   AUDIT GROUP
SEGID = CD                      AUDITED FIELD SEGMENT ID
FIELD = CLOR                    AUDITED FIELD FIELD ID
PRELIM                          PREAUDIT
P1                              PHASE P1
ACCUM = SACDDEC2  *  1.5
CLOR = SACDDEC2 + CLOR
COUNTER1 = ACCUM + COUNTER1
```

Note: COUNTERA through COUNTERE are used internally by the generated operation descriptions to store values as shown in the above example. These counters should not be referenced in the audit language statements. COUNTERF as described in Chapter 5, "Audit Logic Processing" always contains the remainder of the last divide operation. It can be used as an arithmetic operand but not as a result field.

The accumulation counter and the nine auxiliary counters can be reset and/or rounded by the following statements:

```
ACCUM = ZERO              ZERO is a keyword used only
COUNTER1...9 = ZERO       for the reset operation
ROUND ACCUM x             X denotes the number of
ROUND COUNTER1...9 x      decimal positions after the round
```

Example: Assume for this example that SACDDEC2 contains 20.75 and the audited field contains 15.5. The field SACDDEC2 is multiplied by 1.5 and the product (31.125) is placed in the accumulation counter. The value is next rounded to two decimal positions giving 31.13. Then the audited field is added to SACDDEC2 and the sum (36.2) is placed in the audited field. The audited field carries only one decimal position. Counter 1 is reset to zero and finally the value in the accumulation counter is added to auxiliary counter 1 and the sum is placed back in auxiliary counter 1. The final two operations could also be accomplished by a move from ACCUM to COUNTER1.

```
SYSID = SAMP                    SYSTEM ID
AGROUP = MAUD                   AUDIT GROUP
SEGID = CD                      AUDITED FIELD SEGMENT ID
FIELD = CLOR                    AUDITED FIELD FIELD ID
PRELIM                          PREAUDIT
P1                              PHASE P1
ACCUM = SACDDEC2  *  1.5
ROUND ACCUM 2
CLOR = SACDDEC2 + CLOR
COUNTER1 = ZERO
COUNTER1 = ACCUM + COUNTER1
```

STATEMENTS THAT SET INDICATORS AND SWITCHES

Indicator statements are used to set (ON or OFF) flags, bit field values or screen attributes. They are used to set indicators that define fields and segments, control attributes that define screens and set internal system switches. They are coded in the following three formats.

SETTING FIELD INDICATORS AND SCREEN ATTRIBUTES

FORMAT 1a: <field> indicator = (ON,OFF)

Format 1a allows the following indicators to be set for a field

BIT	If a field type is BIT the value can be set to 0 (OFF) or 1 (ON).
CHANGED	Mark a field modified (ON) in the Segment Layout Rule. (OFF) is not allowed.
NULL	Set field indicator variable to NULL (ON). (OFF) is not allowed
CURSOR	Set cursor on the screen to the field (ON) or remove cursor from the field (OFF).
DISP	Set field displayable (ON) or non-displayable (OFF).
HILITE	Set field bright (ON) or normal (OFF).
PREMODIFY	Set field premodified (ON) or normal (OFF).
UPDATE	Set field non-protected (ON) or protected (OFF).

FORMAT 1b: <field> XHILT = (UNDERSCORE,REVERSE,BLINK,DEFAULT)

XHILT 1b sets the high lighting for a field where DEFAULT is no highlighting.

FORMAT 1c: <field> COLOR = (PINK,BLUE,GREEN,RED,WHITE,YELLOW,TURQUOISE)
COLOR sets the color for a field

FORMAT 1d: <field> OUTLINE = value

where value is any combination of the letters O, U, L, R (sets outlining Over, Under, Left, or Right), or the letter N (to turn off all outlining)

The value may be: (1) enclosed in quotes as a literal with no commas, or blanks; or (2) contained in a related field whose length is at least four bytes.

Setting field outline attributes for non-DBCS terminals may not be valid.

The field name specified is the audited or a related field. If FIELD is not specified, the field name defaults to the audited field.

SETTING SEGMENT INDICATORS

FORMAT 2: <'>segid<'> indicator = (ON,OFF)

Format 2 allows the following indicators to be set for a segment

CHGEFLAG	Mark a segment modified (ON) or not modified (OFF).

DLETFLAG — Specified that a segment should be deleted (ON) or change a previous delete specification (OFF). This flag controls deletion at the end of the transaction.

RTRVFLAG — Specifies that a segment is retrieved (ON) or not retrieved (OFF). This flag combined with the changed flag will effect the type of update at the end of the transaction: replace if changed and retrieved; insert if changed and not retrieved.

Segid specifies a two-character data base segment ID.

SETTING SYSTEM INDICATORS

```
FORMAT 3:  SKSDISP = (OFF,STOP)
                or
           SWITCHi = (ON,OFF)
```

SKSDISP — is valid only for the Secondary Key Selection call (KEY-P1) and specifies that the segment occurrence currently being audited should not be displayed or that Secondary Key Selection processing be terminated. Refer to Chapter 5, "Audit Logic Processing" for details on Secondary Key Selection audits.

SWITCHi — specifies an internal switch (i=1-9) which can be set ON or OFF. These switches can be checked with Indicator Expressions as previously discussed. Refer to Chapter 5, "Audit Logic Processing" for details on setting and testing internal switches.

Example: Turn on the bit field SACDBITT. Highlight the audited field on the screen. Turn off the internal switch #3.

```
SACDBITT BIT = ON
CLOR HILITE = ON
SWITCH3 = OFF
```

SYSTEM RELATED STATEMENTS

Two types of statements are used to create calls to the Data Communication system under which IMSADF II runs. They are:

ISSUING A 'ROLL' CALL

An IMS/VS 'ROLL' call can be generated by the following statement.

```
ROLLCALL < <=> 'message text'>
```

The optional message text is inserted to the Express IOPCB prior to the 'ROLL' call. If 'message text' is omitted only the 'ROLL' call is issued.

SECONDARY TRANSACTION STATEMENTS

This category of statements is used to send secondary transactions.

A secondary transaction can be flagged for sending by the following statement:

```
SEND <=> <IMMED> 'itrmodnm'
```

IMMED — specifies that a secondary transaction is to be sent immediately. This allows one ITRMOD definition to be used for multiple secondary transactions.

'itrmodnm' is an eight-character name which equals the ITRMOD name in the Input Transaction Rule and the Output Format Rule name.

A secondary transaction can be directly routed to another IMS/VS System by the following statement:

```
DIRECT 'itrmodnm' to 'msc link'
```

Where 'itrmodnm' is as above and 'msc link' is the Multiple System Couple link name.

CREATING TABLES

DEFINING TABLE LOCATION

Tables can be defined and accessed through the Audit Language. Tables are accessed by ENCODE/DECODE statements which are explained previously in this chapter.

Tables are defined either in an audit routine or by a table definition under a separate table key.

If tables are defined under an audit routine then the standard identifier statements are used along with the Call identifier, TABLES. The generated table segments are placed under the key defined by the routine's identifier statements.

Example: A standard audit is set up in phase P1 under the key 'SAMPGRP1SACDCLOR'. A table is also created under the same key. This technique is usually used when the created audits and tables are combined into a common audit static rule.

```
SYSID  = SAMP
AGROUP = GRP1
SEGID  = CD
FIELD  = CLOR
PROCESS
P1
audit language statements
.
.
.
TABLES
table entries
.
.
.
```

If the table(s) are created under a key different from the audit routines a TBLROOT definition is used. This statement is coded as follows:

```
TBLROOT = '16-char rootname'
```

The TBLROOT value defines the key for the table values. No call or phase identifiers may be coded in this type of table definition. Example: A table is created with the key of 'SAMPLE TABLE #1'.

```
SYSID  = SAMP
AGROUP = GRP1
SEGID  = CD
FIELD  = CLOR
TABLROOT = 'SAMPLE TABLE  #1'
table entries
.
.
.
```

DEFINING TABLE NAME

Multiple tables can be defined under a single TABLE or TBLROOT statement. Each can be defined under a single TABLE or TBLROOT. A table is identified by its name and description. The following two definition statements specify the name (key) of the table and optionally a table description.

```
TBLNAME = 'one- to six-character table name'

TBLDESC = 'one- to 22-character table description'
```

DEFINING TABLE CONTENT

Table entries are coded immediately after the table name and description. An entry is composed of a one- to eight-character argument value followed by a one- to 70-character function value. The argument value is primary key for the table entry and is the source value for a DECODE operation. Its function value is the result. For an ENCODE operation, the function value is the source and its argument value is the result.

Following is the general format for defining a table in an audit routine:

```
TABLES
TBLNAME  = 'TABLE1'
TBLDESC  = 'TABLEDESCRIPTION'
'arg val1' 'function value 1'
'arg val2' 'function value 2'
'arg val3' 'function value 3'
    .          .
    .          .
    .          .
'argvaln' 'functionvalue n'
```

Following is the general format for defining a table under its own root:

```
TBLROOT  = 'TABLEROOTKEYNAME'
TBLNAME  = 'TABLNM'
TBLDESC  = 'TABLE DESCRIPTION'
'arg val1' 'function value 1'
'arg val2' 'function value 2'
'arg val3' 'function value 3'
    .          .
    .          .
    .          .
'argvaln' 'functionvalue n'
```

Notes:

1. Table root identifier, name, description and entries must be enclosed in quotes.

2. Except for the root identifier, which must be exactly 16 characters, a short field is padded to its maximum length by the compiler.

3. A function value may be specified either on the same line as its argument or on the line after.

4. Multiple tables may be specified in one compilation (under same root).

In the following example, two tables are specified.

```
TABLES
* IDENTIFY FIRST TABLE
TBLNAME = 'ABBR 1'                  These two statements
TBLDESC = 'STREET ABBREVIATIONS'    identify the table.
  'AVE '    'AVENUE'                This is a table entry.
  'BLVD'    'BOULEVARD'
    .
    .
```

```
  'STA '    'STATION'
* IDENTIFY SECOND TABLE
TBLNAME = 'STATEC'
TBLDESC = 'STATE CODES'
  'AL'   'ALABAMA'
  'AK'   'ALASKA'
    .
    .
  'WY'   'WYOMING'
```

It is good practice to specify tables in collating sequence of table name, and table entries in collating sequence of argument value.

WRITING AUDIT SUBROUTINES

As described previously a 'CALL' statement can be used to transfer control from an audit routine to an audit subroutine. The subroutine is structured such that it can be called from many routines and will return to the next operation after the call. A subroutine must always reside in the same phase as the calling routine (i.e., calling routine P1 phase, subroutine in P1 phase).

A subroutine differs from a standard audit routine in two ways. First, the 16-character root key under which it resides is not formatted as is that of a standard audit routine. The key is free form and can contain any combination of printable characters. Since the Audit Language combines Field Identifier Statements (SYSID, AGROUP, SEGID, FIELD) to create the key, an additional keyword SUBNAME is required for subroutines. SUBNAME defines the 16-character key of the routine being generated. If SUBNAME is specified the other four keywords are ignored.

The second difference is the use of an audited field. A standard audit routine defines an audited field through the SEGID and FIELD keywords. The following Procedural Statements can refer to the audited field by the 1-4 character FIELD definition. Subroutines at the point they are called take on the characteristics of the calling routine. This means that the audited field of the calling routine becomes the audited field of the subroutine. The PARMNAME keyword allows specification of a one- to four-character name to be considered as the audited field of the subroutine. The user may then use this name when coding statements requiring the audited field.

Two statements (RETURN and GOBACK) are used in a subroutine to return control to the calling routine. They both generate the same operation and may be used interchangeably. The audit may be terminated in a subroutine either through a normal or error condition. If the subroutine terminates with an error (ERRORMSG = xxxx) the audited field of the calling routine is flagged with the error.

Example: A subroutine is created with the key of 'SUBROUTINE_EX.#1'. It can be called from any audit routine executing in a standard audit call under phase P1. When called, it checks the audited field of the calling routine against the field SACDDEC2 and if they are equal the audit is stopped with error code 1111. Then the audited field is compared equal to SACDDEC3 and if equal error 2222 is set. If either error condition exists the audited field of the calling routine is marked in error.

```
SUBNAME = 'SUBROUTINE-EX.#1'
PARMNAME = BASE
PROCESS
    P1
    IF  BASE = SACDDEC2          BASE references audited
        ERRORMSG = 1111          field of calling routine.
    ENDIF
    IF  BASE = SACDDEC3
        ERRORMSG = 2222
    ENDIF
    RETURN
```

CONTROLLING OUTPUT LISTING FORMAT

Listing control and comment statements allow the user to insert comments and provide printer control of the generated batch output listing.

Comment statements consist of a record with an asterisk (*) in column 1, followed by a string of alphanumeric printable characters. They may appear before or after any audit language statement, but cannot be embedded within the statement. All comments found within definition statements are grouped together at the beginning of the generation listing. All other comment statements appear in the listing at the point they are encountered.

Listing control statements are used to control parameters of the output listing. The following keywords are used:

1. LISTING = (YES,NO)

   The default condition is YES. If NO is specified and errors are detected, a listing is produced so that the erroneous statements can be identified.

2. TITLE = 'Maximum of 40-character literal'

   The literal specified in the title statement is printed as part of the header at the top of each page. A new title statement overriding a previous one will take effect when the next page header is printed.

3. LINECOUNT = Number

   The LINECOUNT statement controls the number of lines appearing on a listing page. The page header and two blank lines are included in the page line count. The default value is 60.

4. INDENT = Number

   The INDENT statement controls the number of columns the source code in the listing is indented beneath each IF, DO, or ELSE statement. The default value is 2.

The following two statements may not appear prior to the initial phase identifier. If they are interspersed with the field identifiers, they are ignored.

1. SPACE
   SPACE = 1
   SPACE = 2

   The space statement causes one or two blank lines to be inserted in the listing. If no digit is supplied, SPACE = 1 is assumed.

2. EJECT

   The EJECT statement has no parameters. It causes a skip to a new listing page and the printing of a header on that page.

CONTROLLING SEQUENCE NUMBERING OF AUDIT OPERATIONS

Code generation control statements are used to control the sequence numbers generated for the audit operation descriptors. The following keywords are used:

1. XPANDLBLS = (YES,NO)

   The XPANDLBLS statement increases the number of labels (sequence numbers) available to the user by including the alphabetic characters as valid label components. This enables the user to write larger Auditor programs. The 35 characters A through Z and 1 through 9 are used to generate the two-character labels. The collating sequence is from A to 9 and carry from the first to the second character of the label (i.e., AA....AZ,A1...A9,BA...etc.). In order to avoid the possibility of generating a 00 label, (which specifies end of audit), the character 0 (zero) has been excluded from the list of symbols used to form the 2 character labels. XPANDLBLS = YES will increase the number of valid labels from 99 to a maximum of 1225. The default NO assigns numbers from 01...99.

2. STRTLBL = Two alphanumeric characters

   The STRTLBL statement allows the user to select the label to be assigned to the first Operation Descriptor in each of the P0, P1, and/or P2 legs. The value selected holds for the entire generation but may be changed between routines. If the XPANDLBLS option had been specified previous to this statement, the labels generated will be from the 35 alphanumeric character set described in the XPANDLBLS statement. Otherwise, the two alphanumeric characters must be all numeric. The use of a zero is valid with the numeric only labels. The default starting values are either AA or 01 depending on the XPANDLBLS option.

3. STMTINCR = number

   The STMTINCR statement controls the amount by which each successive sequence number is incremented. The default value is 1.

If multiple generations are executed in one run, all Comment and Control Statements must be placed after the first Definition Statement for the generation to which they apply. If placed before the first Definition Statement they will apply to the previous generation.

AUDIT LANGUAGE ROUTINES

The following limits should be considered when coding audit routines. For this discussion the term 'compilation' means all operations generated in one leg of the Audit Data Base. This means that the combination of preaudit and standard audit within a leg is a 'compilation'.

```
PRELIM
  P0
PROCESS
  P0
```

In the above example the first operation generated in the AFA leg, tests for preaudit (true or false). The next true statement points to the next operation and the next false statement points to the first standard audit operation

```
0116        0205
02    Preaudit
03    operations
04
05    Standard Audit operations
```

CODING LIMITS

1.	Nested IF and DO WHILE	50	
2.	IF and DO WHILE statements in a compilation	400	
3.	Error and Informational messages in a compilation	100	
4.	Explicit GOTO label names in a compilation	100	
5.	Total number of generated operations in a compilation	99	(DEFAULT)
		1225	(XPANDLBLS)

The following is a list of all keywords in the audit language. Related field names cannot match these keywords.

$DATE	DIRECT	GNP	LE	PREMODIFY	STOP
$TIME	DISP	GNPQ	LINECOUNT	PROCESS	STRTLBL
	DLET	GNQ	LISTING	P0	SUBNAME
ACCUM	DLETFLAG	GOBACK	LT	P1	SUBSTR
ADFID	DO	GOTO	LTERM	P2	SWITCH1-9
AEXIT	DOTWIN	GREEN			SYSID
AGROUP		GT	MAP	RED	
ALPHA	EJECT	GU	MASK	REPL	TABLES
ALPHAB	ELSE	GUU	MODE	RETRIEVAL	TBLROOT
ALT	EMBEDB	GU1	MSG	RETURN	TITLE
ANUM	ENCODED			REVERSE	TRUNC
ANUMB	ENDDO	HDEL	NE	ROLLCALL	TRXID
ANYERR	ENDIF	HILITE	NOEMBEDB	ROUND	TURQUOISE
ARG	ENDTWIN	HNPQ	NON$	RTRVFLAG	
	ERROR	HREP	NOP		UPDATE
BIT	ERRORMSG		NOT	SEGID	USERID
BLINK	EQ	IF	NULL	SEND	USING
BLUE	EXIT	IMMED	NUM	SETARRAY	UNDERSCORE
		IN	NUMB	SETERRMSG	
CALL	FIELD	INDENT		SETTWIN	VAL
CHANGE		INFOMSG	OFF	SGN	
CHGEFLAG	GE	INITSEGS	OK	SKSDISP	WHITE
COLOR	GHN	ISRT	ON	SPACE	WARNMSG
COMMON	GHNP		OUTLINE	SPAERMSG	WHILE
CONCAT	GNHQ	KANAME		SPAKEYID	
COPYSEG	GHU	KEY	PARMNAME	SPASQL	XHILT
COUNTER	GHUU	KEYFIELD	PASS	SPAULANG	XPANDLBLS
COUNTER	GN	KSELECT3-9	PCBNUM	SPAWHERE	
CURSOR			PGROUP	SQL	YELLOW
			PINK	SQLCODE	
DBCS			PRELIM	SQLWARN0-7	ZERO
DEFAULT				STATCODE	
DECODED				STD	
				STMTINCR	

USING THE AUDIT LANGUAGE PROCEDURE

A procedure is provided to invoke the Audit Language Processor. The procedure is distributed with the name ????AL, where ???? will be replaced with the IMSADF II System ID at installation. The procedure consists of two steps which generate the batch input stream and update the Audit Data Base. An input DD statement with the name AUDGEN.INDATA must be supplied pointing to the sequential Audit Language Source input stream. If errors occur in the generation (Step 1), the data base update (Step 2) will not occur. If a data base update is not desired at the time of program generation, the output data set (OUTDATA) can be changed to point to a punch file (SYSOUT=B) or a permanent data set and the second step deleted. If OUTDATA is SYSOUT=B the output is punched whether or not an error occurs. If OUTDATA is a data set it is closed, reopened, and closed, eliminating the output if an error occurs.

Creating static audit rules from Audit Language source statements is an option that the Audit Language Processor provides. If static rules are created, the data base will not be updated (Step 2). Static audit rules can be created by changing the PARM parameter in the Audit Language Processor JCL to PARM='STAT'. The AUDGEN.INDATA DD statement, as described above, must exist and the output data set (OUTDATA) should be a permanent or temporary data set. It should not be SYSOUT=B. An output data set (MSGOUT) provides a listing of the static audit rules and any error messages. Another data set (STADOUT) is the output data set that can be used as input to the Rules Generator to compile and link-edit the rule. The DD statement (STADOUT) should be modified from SYSOUT=B to a sequential file on tape or disk. The file will be presented as SYSIN to the Rules Generator. Refer to the heading 'Static Audit Rules' in Appendix D of the IMS Application Development Facility II Version 2 Release 2 Application Development Guide for more information on invoking the Rules Generator and restrictions that apply when creating static audit rules.

AUDIT LANGUAGE EXAMPLE

Figure 3-2 and Figure 3-3 contain an example of an Audit language generation demonstrating use of various Audit language operands. It does not represent any specific logic.

Figure 3-2 shows the generation listing from step 1 of the procedure, and Figure 3-3 shows the batch Audit Data Base updates processed in step 2.

```
   HIGH LEVEL AUDIT LANGUAGE LISTING                'AUDIT LANGUAGE EXAMPLES'

                                          0001       ADFID   = MFC1
                                          0002       SYSID   = SAXX
                                          0003       AGROUP  = TEST
                                          0004       SEGID   = CD
MFC1B3GPSAXXTESTSACDCLOR                             FIELD   = CLOR
MFC1B4GPSAXXTESTSACDCLOR
                                          0005       PRELIM
AA 01 16         0209                     0006       P0
AA 02 C1         03   0900                0007       IF SACDDEC1 = SACDDEC2
DA 02 0001(SACDDEC2,SACDDEC1)
AA 03 C0         0404                     0008         SACDDEC3 = SACDDEC4
DA 03 0001(SACDDEC4,SACDDEC3)
                                          0009       ELSE
   02                                     0010         ERRORMSG = 0900
                                          0011       ENDIF
AA 04 06SACDDEC20506                      0012       IF SACDDEC2 GT 10
DA 04 0001(10)
AA 05 11SACDDEC10000                      0013         SACDDEC1 = CLOR
                                          0014       ELSE
AA 06 07SACDDEC20700                      0015         DO WHILE SACDDEC2 < 10
DA 06 0001(10)
AA 07 C4SACDDEC10808                      0016           SACDDEC1 = SACDDEC1 + CLOR
DA 07 0001(SACDDEC1,SACDCLOR)
AA 08 C4SACDDEC20606                      0017           SACDDEC2 = SACDDEC2 + 1
DA 08 0001(SACDDEC2,1)
                                          0018         ENDDO
                                          0019       ENDIF
FA 01 16         0208                     0020       P1
FA 02 01         0303                     0021       NOP
FA 03 24SACDDEC20406                      0022       IF SACDDEC2 = NUM
FA 04 43SACDDEC20506                      0023         IF SACDDEC2 = NON$ '$$12$,$$13$'
DF 04 0001($$12$,$$13$)
FA 05 12         0606                     0024           CLOR = 10
DF 05 0001(10)
                                          0025         ENDIF
                                          0026       ENDIF
FA 06 B2SACDVALU0700                      0027       IF ENCODED CLOR TO SACDVALU USING 'TABLECODES      TABLE1' OK
DF 06 0001(TABLECODES        TABLE1)
FA 07 B6SACDDEC200   9999                 0028         IF SACDDEC2 NOT IN ARG 'TABLECODES      TABLE1'
DF 07 0001(TABLECODES        TABLE1)
   07                                     0029           ERRORMSG 9999
                                          0030         ENDIF
                                          0031       ENDIF
MA 01 16         0204                     0032       P2
MA 02 75         0300                     0033       IF AEXIT 75 RETURN = TRUE
MA 03 W2SACDDEC200   7777                 0034         IF AEXIT W2 SACDDEC2 RETURN EQ FALSE
   03                                     0035           INFOMSG = 7777
                                          0036         ENDIF
                                          0037       ENDIF
                                          0038       PROCESS
                                          0039       P0
AA 09 C1         1011                     0040       IF SACDDEC1 = SACDDEC2
DA 09 0001(SACDDEC2,SACDDEC1)
AA 10 C0         1313                     0041         SACDDEC3 = SACDDEC4
DA 10 0001(SACDDEC4,SACDDEC3)
                                          0042       ELSE
```

Figure 3-2 (Part 1 of 3). Audit Language Generation Listing

```
AA 11 C0          1212                0043          SACDDEC3 = SACDDEC5
DA 11 0001(SACDDEC5,SACDDEC3)
AA 12 01          1515                0044          GOTO COMPUTE1
                                      0045        ENDIF
AA 13 C4SACDDEC11414                  0046        SACDDEC1 = SACDDEC1 + 1
DA 13 0001(SACDDEC1,1)
AA 14 C4SACDDEC21515                  0047        SACDDEC2 = SACDDEC2 + 1
DA 14 0001(SACDDEC2,1)
AA 15 10SACDDEC11616                  0048COMPUTE1: CLOR = SACDDEC1
AA 16 49          1717                0049        ACCUM = SACDDEC2 * 1.5
AA 17 51SACDDEC21818
AA 18 58          1919
DA 18 0001(1.5)
AA 19 C4SACDCLOR2020                  0050        CLOR = SACDDEC2 + CLOR
DA 19 0001(SACDDEC2,SACDCLOR)
AA 20 63COUNTERA2121                  0051        COUNTER1 = ACCUM + COUNTER1
AA 21 C4COUNTER12222
DA 21 0001(COUNTERA,COUNTER1)
AA 22 C4COUNTER10000                  0052        COUNTER1 = CLOR
DA 22 0001(SACDCLOR,0)
                                      0053        P1
FA 08 D0          0911                0054        IF RETRIEVAL = 'A1,A2'
DF 08 0001(A1,A2)
FA 09 67          1011                0055          IF MODE = 4
DF 09 0001(4)
FA 10 12          1111                0056            CLOR = 10
DF 10 0001(10)
                                      0057          ENDIF
                                      0058        ENDIF
FA 11 17SACDBITT  12 7777             0059        IF SACDBITT BIT = ON
   11                                 0060          ERRORMSG = 7777
                                      0061        ENDIF
FA 12 47SACDDEC213   8888             0062        IF SACDDEC2 ERROR = OFF
   12                                 0063          ERRORMSG = 8888
                                      0064        ENDIF
FA 13 28SWITCH5   14 9999             0065        IF SWITCH5 = ON
   13                                 0066          ERRORMSG = 9999
                                      0067        ENDIF
FA 14 75          1516                0068        IF AEXIT 75 RETURN = TRUE
FA 15 W2SACDDEC216   7777             0069          IF AEXIT W2 SACDDEC2 RETURN EQ FALSE
   15                                 0070            ERRORMSG = 7777
                                      0071          ENDIF
                                      0072        ENDIF
FA 16 18SACDBITT1717                  0073        SACDBITT BIT = ON
FA 17 A9          1818                0074        CLOR HILITE = ON
DF 17 0001(SACDCLOR,HILT,1)
FA 18 27SWITCH3 1919                  0075        SWITCH3 = OFF
FA 19 C1          2021                0076        IF SACDDEC1 = SACDDEC2
DF 19 0001(SACDDEC2,SACDDEC1)
FA 20 C0          2121                0077          SACDDEC3 = SACDDEC4
DF 20 0001(SACDDEC4,SACDDEC3)
                                      0078        ENDIF
FA 21 21          2223                0079        IF CLOR IN 100:200,500:600
DF 21 0001(100,200)
DF 21 0002(500,600)
FA 22 12          2323                0080          CLOR = 1000
DF 22 0001(1000)
```

Figure 3-2 (Part 2 of 3). Audit Language Generation Listing

```
HIGH LEVEL AUDIT LANGUAGE LISTING                    'AUDIT LANGUAGE EXAMPLES'

                                          0081        ENDIF
FA 23 32SACDCODE2524                      0082        IF SACDCODE NOT IN 'AB,AC,AD'
DF 23 0001(AB,AC,AD)
FA 24 09SACDCODE2525                      0083          SACDCODE = 'DD'
DF 24 0001(DD)
                                          0084        ENDIF
FA 25 36SACDKEY12627                      0085        IF GU SACDKEY1 AA OK
DF 25 0001(AA,GU  )
FA 26 09SACDDEC10000                      0086          SACDDEC1 = 1
DF 26 0001(1)
                                          0087        ELSE
FA 27 37        2800                      0088          IF STATCODE = 'GE'
DF 27 0001(GE)
FA 28 36SACDKEY12930                      0089            IF ISRT IMMED SACDKEY1 AA OK
DF 28 0001(AA,ISRT,I)
FA 29 09SACDDEC10000                      0090              SACDDEC1 = 2
DF 29 0001(2)
                                          0091            ELSE
FA 30 09SACDDEC10000                      0092              SACDDEC1 = 3
DF 30 0001(3)
                                          0093            ENDIF
                                          0094          ENDIF
                                          0095        ENDIF
                                          0096        P2
MA 04 75        0506                      0097        IF AEXIT 75 RETURN = TRUE
MA 05 W2SACDDEC206  7777                  0098          IF AEXIT W2 SACDDEC2 RETURN EQ FALSE
   05                                     0099            INFOMSG = 7777
                                          0100          ENDIF
                                          0101        ENDIF
MA 06 30SACDDEC1  07 0010                 0102        IF CLOR = SACDDEC1
   06                                     0103          INFOMSG = 0010
                                          0104        ENDIF
MA 07 30SACDDEC2  00 0020                 0105        IF CLOR = SACDDEC2
   07                                     0106          INFOMSG = 0020
                                          0107        ENDIF
```

Figure 3-2 (Part 3 of 3). Audit Language Generation Listing

```
PAGE 000001                    IMS APPLICATION DEVELOPMENT FACILITY DATA BASE PROCESSING
TIME = 1033522          ----+----1----+----2----+----3----+----4----+----5----+----6----+----7----+----8

          RECORD # 00001  MFC1B3GFSAXXTESTSACDCLOR
     TRANSACTION # 00001  *** SEGMENT DELETED SUCCESSFULLY ***

          RECORD # 00002  MFC1B4GFSAXXTESTSACDCLOR
     TRANSACTION # 00002  *** SEGMENT ADDED SUCCESSFULLY ***

          RECORD # 00003  MFC1B4AASAXXTESTSACDCLOR0116        0209
     TRANSACTION # 00003  *** SEGMENT ADDED SUCCESSFULLY ***

          RECORD # 00004  MFC1B4AASAXXTESTSACDCLOR02C1        03  0900
     TRANSACTION # 00004  *** SEGMENT ADDED SUCCESSFULLY ***

          RECORD # 00005  MFC1B4DASAXXTESTSACDCLOR020001(SACDDEC2,SACDDEC1)
     TRANSACTION # 00005  *** SEGMENT ADDED SUCCESSFULLY ***

          RECORD # 00006  MFC1B4AASAXXTESTSACDCLOR03C0        0404
     TRANSACTION # 00006  *** SEGMENT ADDED SUCCESSFULLY ***

          RECORD # 00007  MFC1B4DASAXXTESTSACDCLOR030001(SACDDEC4,SACDDEC3)
     TRANSACTION # 00007  *** SEGMENT ADDED SUCCESSFULLY ***

          RECORD # 00008  MFC1B4AASAXXTESTSACDCLOR0406SACDDEC20506
     TRANSACTION # 00008  *** SEGMENT ADDED SUCCESSFULLY ***

          RECORD # 00009  MFC1B4DASAXXTESTSACDCLOR040001(10)
     TRANSACTION # 00009  *** SEGMENT ADDED SUCCESSFULLY ***

          RECORD # 00010  MFC1B4AASAXXTESTSACDCLOR0511SACDDEC10000
     TRANSACTION # 00010  *** SEGMENT ADDED SUCCESSFULLY ***

          RECORD # 00011  MFC1B4AASAXXTESTSACDCLOR0607SACDDEC20700
     TRANSACTION # 00011  *** SEGMENT ADDED SUCCESSFULLY ***

          RECORD # 00012  MFC1B4DASAXXTESTSACDCLOR060001(10)
     TRANSACTION # 00012  *** SEGMENT ADDED SUCCESSFULLY ***

          RECORD # 00013  MFC1B4AASAXXTESTSACDCLOR07C4SACDDEC10808
     TRANSACTION # 00013  *** SEGMENT ADDED SUCCESSFULLY ***

          RECORD # 00014  MFC1B4DASAXXTESTSACDCLOR070001(SACDDEC1,SACDCLOR)
     TRANSACTION # 00014  *** SEGMENT ADDED SUCCESSFULLY ***

          RECORD # 00015  MFC1B4AASAXXTESTSACDCLOR08C4SACDDEC20606
     TRANSACTION # 00015  *** SEGMENT ADDED SUCCESSFULLY ***

          RECORD # 00016  MFC1B4DASAXXTESTSACDCLOR080001(SACDDEC2,1)
     TRANSACTION # 00016  *** SEGMENT ADDED SUCCESSFULLY ***

          RECORD # 00017  MFC1B4AASAXXTESTSACDCLOR09C1        1011
     TRANSACTION # 00017  *** SEGMENT ADDED SUCCESSFULLY ***

          RECORD # 00018  MFC1B4DASAXXTESTSACDCLOR090001(SACDDEC2,SACDDEC1)
```

Figure 3-3 (Part 1 of 6). Audit Data Base Update Created by Audit Language Generation

```
PAGE 000002                          IMS APPLICATION DEVELOPMENT FACILITY DATA BASE PROCESSING
TIME = 1033545                 ----+----1----+----2----+----3----+----4----+----5----+----6----+----7----+----8
          TRANSACTION # 00018  *** SEGMENT ADDED SUCCESSFULLY ***

               RECORD # 00019  MFC1B4AASAXXTESTSACDCLOB10C0          1313
          TRANSACTION # 00019  *** SEGMENT ADDED SUCCESSFULLY ***

               RECORD # 00020  MFC1B4DASAXXTESTSACDCLOB100001(SACDDEC4,SACDDEC3)
          TRANSACTION # 00020  *** SEGMENT ADDED SUCCESSFULLY ***

               RECORD # 00021  MFC1B4AASAXXTESTSACDCLOB11C0          1212
          TRANSACTION # 00021  *** SEGMENT ADDED SUCCESSFULLY ***

               RECORD # 00022  MFC1B4DASAXXTESTSACDCLOB110001(SACDDEC5,SACDDEC3)
          TRANSACTION # 00022  *** SEGMENT ADDED SUCCESSFULLY ***

               RECORD # 00023  MFC1B4AASAXXTESTSACDCLOB1201          1515
          TRANSACTION # 00023  *** SEGMENT ADDED SUCCESSFULLY ***

               RECORD # 00024  MFC1B4AASAXXTESTSACDCLOB13C4SACDDEC11414
          TRANSACTION # 00024  *** SEGMENT ADDED SUCCESSFULLY ***

               RECORD # 00025  MFC1B4DASAXXTESTSACDCLOB130001(SACDDEC1,1)
          TRANSACTION # 00025  *** SEGMENT ADDED SUCCESSFULLY ***

               RECORD # 00026  MFC1B4AASAXXTESTSACDCLOB14C4SACDDEC21515
          TRANSACTION # 00026  *** SEGMENT ADDED SUCCESSFULLY ***

               RECORD # 00027  MFC1B4DASAXXTESTSACDCLOB140001(SACDDEC2,1)
          TRANSACTION # 00027  *** SEGMENT ADDED SUCCESSFULLY ***

               RECORD # 00028  MFC1B4AASAXXTESTSACDCLOB1510SACDDEC11616
          TRANSACTION # 00028  *** SEGMENT ADDED SUCCESSFULLY ***

               RECORD # 00029  MFC1B4AASAXXTESTSACDCLOB1649          1717
          TRANSACTION # 00029  *** SEGMENT ADDED SUCCESSFULLY ***

               RECORD # 00030  MFC1B4AASAXXTESTSACDCLOB1751SACDDEC21818
          TRANSACTION # 00030  *** SEGMENT ADDED SUCCESSFULLY ***

               RECORD # 00031  MFC1B4AASAXXTESTSACDCLOB1858          1919
          TRANSACTION # 00031  *** SEGMENT ADDED SUCCESSFULLY ***

               RECORD # 00032  MFC1B4DASAXXTESTSACDCLOB180001(1,5)
          TRANSACTION # 00032  *** SEGMENT ADDED SUCCESSFULLY ***

               RECORD # 00033  MFC1B4AASAXXTESTSACDCLOB19C4SACDCLOB2020
          TRANSACTION # 00033  *** SEGMENT ADDED SUCCESSFULLY ***

               RECORD # 00034  MFC1B4DASAXXTESTSACDCLOB190001(SACDDEC2,SACDCLOB)
          TRANSACTION # 00034  *** SEGMENT ADDED SUCCESSFULLY ***

               RECORD # 00035  MFC1B4AASAXXTESTSACDCLOB2063COUNTERA2121
          TRANSACTION # 00035  *** SEGMENT ADDED SUCCESSFULLY ***

               RECORD # 00036  MFC1B4AASAXXTESTSACDCLOB21C4COUNTER12222
```

Figure 3-3 (Part 2 of 6). Audit Data Base Update Created by Audit Language Generation

```
PAGE 000003                          IBS APPLICATION DEVELOPMENT FACILITY DATA BASE PROCESSING
TIME = 1033558                       ----+----1----+----2----+----3----+----4----+----5----+----6----+----7----+----8
          TRANSACTION # 00036  *** SEGMENT ADDED SUCCESSFULLY ***

               RECORD # 00037  MFC1B4DASAIXTESTSACDCLOR210001(COUNTERA,COUNTER1)
          TRANSACTION # 00037  *** SEGMENT ADDED SUCCESSFULLY ***

               RECORD # 00038  MFC1B4AASAIXTESTSACDCLOR22C4COUNTER10000
          TRANSACTION # 00038  *** SEGMENT ADDED SUCCESSFULLY ***

               RECORD # 00039  MFC1B4DASAIXTESTSACDCLOR220001(SACDCLOR,0)
          TRANSACTION # 00039  *** SEGMENT ADDED SUCCESSFULLY ***

               RECORD # 00040  MFC1B4FASAIXTESTSACDCLOR0116        0208
          TRANSACTION # 00040  *** SEGMENT ADDED SUCCESSFULLY ***

               RECORD # 00041  MFC1B4FASAIXTESTSACDCLOR0201        0303
          TRANSACTION # 00041  *** SEGMENT ADDED SUCCESSFULLY ***

               RECORD # 00042  MFC1B4FASAIXTESTSACDCLOR0324SACDDEC20406
          TRANSACTION # 00042  *** SEGMENT ADDED SUCCESSFULLY ***

               RECORD # 00043  MFC1B4FASAIXTESTSACDCLOR0443SACDDEC20506
          TRANSACTION # 00043  *** SEGMENT ADDED SUCCESSFULLY ***

               RECORD # 00044  MFC1B4DFSAIXTESTSACDCLOR040001($$12$,$$13$)
          TRANSACTION # 00044  *** SEGMENT ADDED SUCCESSFULLY ***

               RECORD # 00045  MFC1B4FASAIXTESTSACDCLOR0512        0606
          TRANSACTION # 00045  *** SEGMENT ADDED SUCCESSFULLY ***

               RECORD # 00046  MFC1B4DFSAIXTESTSACDCLOR050001(10)
          TRANSACTION # 00046  *** SEGMENT ADDED SUCCESSFULLY ***

               RECORD # 00047  MFC1B4FASAIXTESTSACDCLOR06B2SACDVALU0700
          TRANSACTION # 00047  *** SEGMENT ADDED SUCCESSFULLY ***

               RECORD # 00048  MFC1B4DFSAIXTESTSACDCLOR060001(TABLECODES      TABLE1)
          TRANSACTION # 00048  *** SEGMENT ADDED SUCCESSFULLY ***

               RECORD # 00049  MFC1B4FASAIXTESTSACDCLOR07B6SACDDEC200  9999
          TRANSACTION # 00049  *** SEGMENT ADDED SUCCESSFULLY ***

               RECORD # 00050  MFC1B4DFSAIXTESTSACDCLOR070001(TABLECODES      TABLE1)
          TRANSACTION # 00050  *** SEGMENT ADDED SUCCESSFULLY ***

               RECORD # 00051  MFC1B4FASAIXTESTSACDCLOR08D0        0911
          TRANSACTION # 00051  *** SEGMENT ADDED SUCCESSFULLY ***

               RECORD # 00052  MFC1B4DFSAIXTESTSACDCLOR080001(A1,A2)
          TRANSACTION # 00052  *** SEGMENT ADDED SUCCESSFULLY ***

               RECORD # 00053  MFC1B4FASAIXTESTSACDCLOR0967        1011
          TRANSACTION # 00053  *** SEGMENT ADDED SUCCESSFULLY ***

               RECORD # 00054  MFC1B4DFSAIXTESTSACDCLOR090001(4)
```

Figure 3-3 (Part 3 of 6). Audit Data Base Update Created by Audit Language Generation

```
PAGE 000004                    IMS APPLICATION DEVELOPMENT FACILITY DATA BASE PROCESSING
TIME = 1033577                 ----+----1----+----2----+----3----+----4----+----5----+----6----+----7----+----8
          TRANSACTION # 00054  *** SEGMENT ADDED SUCCESSFULLY ***

               RECORD # 00055  MFC1B4FASAIXTESTSACDCLOB1012          1111
          TRANSACTION # 00055  *** SEGMENT ADDED SUCCESSFULLY ***

               RECORD # 00056  MFC1B4DFSAIXTESTSACDCLOB100001(10)
          TRANSACTION # 00056  *** SEGMENT ADDED SUCCESSFULLY ***

               RECORD # 00057  MFC1B4FASAIXTESTSACDCLOB1117SACDBITT  127777
          TRANSACTION # 00057  *** SEGMENT ADDED SUCCESSFULLY ***

               RECORD # 00058  MFC1B4FASAIXTESTSACDCLOB1247SACDDEC213  8888
          TRANSACTION # 00058  *** SEGMENT ADDED SUCCESSFULLY ***

               RECORD # 00059  MFC1B4FASAIXTESTSACDCLOB1328SWITCH5   149999
          TRANSACTION # 00059  *** SEGMENT ADDED SUCCESSFULLY ***

               RECORD # 00060  MFC1B4FASAIXTESTSACDCLOB1475          1516
          TRANSACTION # 00060  *** SEGMENT ADDED SUCCESSFULLY ***

               RECORD # 00061  MFC1B4FASAIXTESTSACDCLOB15W2SACDDEC216  7777
          TRANSACTION # 00061  *** SEGMENT ADDED SUCCESSFULLY ***

               RECORD # 00062  MFC1B4FASAIXTESTSACDCLOB1618SACDBITT1717
          TRANSACTION # 00062  *** SEGMENT ADDED SUCCESSFULLY ***

               RECORD # 00063  MFC1B4FASAIXTESTSACDCLOB17A9          1818
          TRANSACTION # 00063  *** SEGMENT ADDED SUCCESSFULLY ***

               RECORD # 00064  MFC1B4DFSAIXTESTSACDCLOB170001(SACDCLOB,HILT,1)
          TRANSACTION # 00064  *** SEGMENT ADDED SUCCESSFULLY ***

               RECORD # 00065  MFC1B4FASAIXTESTSACDCLOB1827SWITCH3 1919
          TRANSACTION # 00065  *** SEGMENT ADDED SUCCESSFULLY ***

               RECORD # 00066  MFC1B4FASAIXTESTSACDCLOB19C1          2021
          TRANSACTION # 00066  *** SEGMENT ADDED SUCCESSFULLY ***

               RECORD # 00067  MFC1B4DFSAIXTESTSACDCLOB190001(SACDDEC2,SACDDEC1)
          TRANSACTION # 00067  *** SEGMENT ADDED SUCCESSFULLY ***

               RECORD # 00068  MFC1B4FASAIXTESTSACDCLOB20C0          2121
          TRANSACTION # 00068  *** SEGMENT ADDED SUCCESSFULLY ***

               RECORD # 00069  MFC1B4DFSAIXTESTSACDCLOB200001(SACDDEC4,SACDDEC3)
          TRANSACTION # 00069  *** SEGMENT ADDED SUCCESSFULLY ***

               RECORD # 00070  MFC1B4FASAIXTESTSACDCLOB2121          2223
          TRANSACTION # 00070  *** SEGMENT ADDED SUCCESSFULLY ***

               RECORD # 00071  MFC1B4DFSAIXTESTSACDCLOB210001(100,200)
          TRANSACTION # 00071  *** SEGMENT ADDED SUCCESSFULLY ***

               RECORD # 00072  MFC1B4DFSAIXTESTSACDCLOB210002(500,600)
```

Figure 3-3 (Part 4 of 6). Audit Data Base Update Created by Audit Language Generation

```
PAGE 000005                          IBS APPLICATION DEVELOPMENT FACILITY DATA BASE PROCESSING
TIME = 1033596                       ----+----1----+----2----+----3----+----4----+----5----+----6----+----7----+----8
          TRANSACTION # 00072  *** SEGMENT ADDED SUCCESSFULLY ***

               RECORD # 00073  MFC1B4FASAXXTESTSACDCLOR2212          2323
          TRANSACTION # 00073  *** SEGMENT ADDED SUCCESSFULLY ***

               RECORD # 00074  MFC1B4DFSAXXTESTSACDCLOR220001(1000)
          TRANSACTION # 00074  *** SEGMENT ADDED SUCCESSFULLY ***

               RECORD # 00075  MFC1B4FASAXXTESTSACDCLOR2332SACDCODE2524
          TRANSACTION # 00075  *** SEGMENT ADDED SUCCESSFULLY ***

               RECORD # 00076  MFC1B4DFSAXXTESTSACDCLOR230001(AB,AC,AD)
          TRANSACTION # 00076  *** SEGMENT ADDED SUCCESSFULLY ***

               RECORD # 00077  MFC1B4FASAXXTESTSACDCLOR2409SACDCODE2525
          TRANSACTION # 00077  *** SEGMENT ADDED SUCCESSFULLY ***

               RECORD # 00078  MFC1B4DFSAXXTESTSACDCLOR240001(DD)
          TRANSACTION # 00078  *** SEGMENT ADDED SUCCESSFULLY ***

               RECORD # 00079  MFC1B4FASAXXTESTSACDCLOR2536SACDKEY12627
          TRANSACTION # 00079  *** SEGMENT ADDED SUCCESSFULLY ***

               RECORD # 00080  MFC1B4DFSAXXTESTSACDCLOR250001(AA,GU   )
          TRANSACTION # 00080  *** SEGMENT ADDED SUCCESSFULLY ***

               RECORD # 00081  MFC1B4FASAXXTESTSACDCLOR2609SACDDEC10000
          TRANSACTION # 00081  *** SEGMENT ADDED SUCCESSFULLY ***

               RECORD # 00082  MFC1B4DFSAXXTESTSACDCLOR260001(1)
          TRANSACTION # 00082  *** SEGMENT ADDED SUCCESSFULLY ***

               RECORD # 00083  MFC1B4FASAXXTESTSACDCLOR2737          2800
          TRANSACTION # 00083  *** SEGMENT ADDED SUCCESSFULLY ***

               RECORD # 00084  MFC1B4DFSAXXTESTSACDCLOR270001(GE)
          TRANSACTION # 00084  *** SEGMENT ADDED SUCCESSFULLY ***

               RECORD # 00085  MFC1B4FASAXXTESTSACDCLOR2836SACDKEY12930
          TRANSACTION # 00085  *** SEGMENT ADDED SUCCESSFULLY ***

               RECORD # 00086  MFC1B4DFSAXXTESTSACDCLOR280001(AA,ISRT,I)
          TRANSACTION # 00086  *** SEGMENT ADDED SUCCESSFULLY ***

               RECORD # 00087  MFC1B4FASAXXTESTSACDCLOR2909SACDDEC10000
          TRANSACTION # 00087  *** SEGMENT ADDED SUCCESSFULLY ***

               RECORD # 00088  MFC1B4DFSAXXTESTSACDCLOR290001(2)
          TRANSACTION # 00088  *** SEGMENT ADDED SUCCESSFULLY ***

               RECORD # 00089  MFC1B4FASAXXTESTSACDCLOR3009SACDDEC10000
          TRANSACTION # 00089  *** SEGMENT ADDED SUCCESSFULLY ***

               RECORD # 00090  MFC1B4DFSAXXTESTSACDCLOR300001(3)
```

Figure 3-3 (Part 5 of 6). Audit Data Base Update Created by Audit Language Generation

```
PAGE 000006                          IMS APPLICATION DEVELOPMENT FACILITY DATA BASE PROCESSING
TIME = 1034020                       ----+----1----+----2----+----3----+----4----+----5----+----6----+----7----+----8
          TRANSACTION # 00090        *** SEGMENT ADDED SUCCESSFULLY ***

               RECORD # 00091        MFC1B4MASAXXTESTSACDCLOB0116        0204
          TRANSACTION # 00091        *** SEGMENT ADDED SUCCESSFULLY ***

               RECORD # 00092        MFC1B4MASAXXTESTSACDCLOB0275        0300
          TRANSACTION # 00092        *** SEGMENT ADDED SUCCESSFULLY ***

               RECORD # 00093        MFC1B4MASAXXTESTSACDCLOB03W2SACDDEC200  7777
          TRANSACTION # 00093        *** SEGMENT ADDED SUCCESSFULLY ***

               RECORD # 00094        MFC1B4MASAXXTESTSACDCLOB0475        0506
          TRANSACTION # 00094        *** SEGMENT ADDED SUCCESSFULLY ***

               RECORD # 00095        MFC1B4MASAXXTESTSACDCLOB05W2SACDDEC206  7777
          TRANSACTION # 00095        *** SEGMENT ADDED SUCCESSFULLY ***

               RECORD # 00096        MFC1B4MASAXXTESTSACDCLOB0630SACDDEC1  070010
          TRANSACTION # 00096        *** SEGMENT ADDED SUCCESSFULLY ***

               RECORD # 00097        MFC1B4MASAXXTESTSACDCLOB07J0SACDDEC2  000020
          TRANSACTION # 00097        *** SEGMENT ADDED SUCCESSFULLY ***
                                     *** END OF INPUT ***
```

Figure 3-3 (Part 6 of 6). Audit Data Base Update Created by Audit Language Generation

CHAPTER 4. DYNAMIC RULES DATA BASES

There are three data bases used by the system to maintain dynamic rules. They are:

Sign-on Profile Data Base

Message Data Base

Audit Data Base

These data bases can be either a DL/I hierarchical or a DB2 relational structure. The type of data bases used is defined at system installation. These data bases must be all DL/I or all DB2.

This chapter covers the following areas for each data base:

Purpose of the data base

Data base layout

--- DL/I

--- DB2

Updating the data bases

--- DL/I

--- DB2

Under CICS/OS/VS, you display the Sign-on screen by entering:

```
MFC1
```

Under IMS/VS, you display the Sign-on screen by entering:

```
/FORMAT MFC1
```

When the Data Base Administrator Sign-on screen is displayed, enter:

```
999999   (for USERID field)
     X   (for PROJECT field)
     X   (for GROUP field)
```

You are now ready to enter selections from the Primary Option Menu to update the Dynamic Rules Data Bases.

It is always good practice from a security standpoint to add additional profiles for the DBA transaction (SYSID = ????) and eventually to delete the '999999' userid. (It is a good idea to leave '999999' as the SAMP userid, so that anyone can use it).

SIGN-ON PROFILE DATA BASE

The Sign-on Profile Data Base contains data that enables the execution system to perform the following functions:

1. Validate a user's sign-on

2. Tailor the Secondary Option Menu screen for a terminal user

3. Monitor user selected modes of operation (delete, initiate, update, etc.)

4. Collect messages destined for a Project/Group

The validation of a user's sign-on is performed by verifying that the Project/Group and userid entered by the user exist in the Sign-on

Profile Data Base. This validation is performed at the beginning of all conversational sign-ons and optionally during the execution of a batch update. The existence of Project/Group and userid are verified through calls to the Sign-on Profile Data Base. The Project/Group and userid are used as key values. Calls are made to retrieve the Project/Group record, and the User Identification record.

SIGN-ON PROFILE DATA BASE (DL/I VERSION)

The following sections describe the DL/I version of the Sign-on Profile data base.

The Project/Group (PG) segment contains the four-character application system identification. In conversational processing, this identification must match the MOD name entered in the /FOR xxxx command. If it does not, the sign-on is invalid. In batch processing, the four-character application system ID in the PG segment must match the first four characters of the batch transaction driver. If not, the sign-on is invalid.

The Profile Identification (PR) segments are dependents of the PG segment. They are keyed by a two-character profile ID and contain a list of transaction selections and authorized modes of operation.

The number of PR segments under a PG segment will vary in order to contain all transaction IDs for which a user is authorized. Each PR segment can accommodate 75 IDs, and a maximum of four PR segments can exist. The supplied rules relieve the user of maintaining the IDs on a segment-by-segment basis (i.e. the entire list of transaction IDs can be entered on the screen, and the rules will dictate the number of segments accessed). Performance may be improved by keeping the most active transaction IDs within the first 75 IDs of the total list.

Each User Identification (SR) segment has a profile ID field that associates the employee with the appropriate PR segment. In Conversational processing, a Secondary Option Menu screen is tailored for each user by comparing data contained in the Secondary Option Menu Rule with that contained in the Profile Identification (PR) segment for that user in the Sign-on Profile Data Base. The purpose of this comparison is to identify which transactions can be selected by the user. Those transactions that appear in both the Secondary Option Menu Rule and in the PR segments are displayed on the Secondary Option Menu screen. In this way, only those selections that are valid for the user are presented.

In batch processing, the transaction specified in the transaction input must appear in the user's PR segment. If it doesn't, the input transaction is invalid.

The transaction mode entered by the user is also validated against data in the PR segment. Each transaction listed in the PR segment is assigned an authorized mode of operation (1 to 6). The transaction mode specified in the PR segment is the highest level for which a user is eligible. Transaction mode 1 (DELETE) is the highest level possible. Each level includes all lower levels.

The Sign-on Profile Data Base also provides a message (MG) segment in which messages destined for a Project/Group are collected. These messages can be initiated by a user through the Project/Group Message Sending function or can be generated automatically as a result of data base segment deletion or modification. The MG segment is a dependent of the PG segment and, therefore, is associated with a particular Project/Group.

Sign-on Profile Data Base Layout (DL/I Version)

The Sign-on Profile Data Base is a two level HDAM data base. Figure 4-1 illustrates the hierarchical structure of this data base.

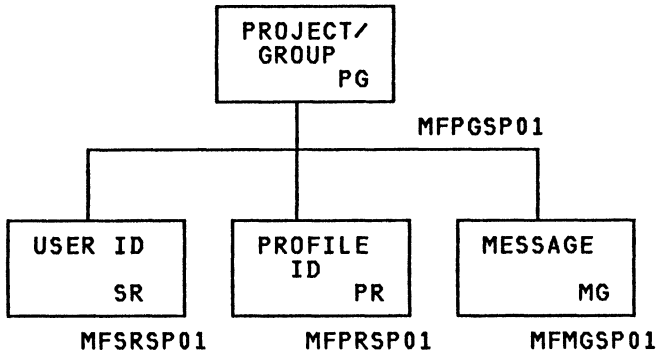


Figure 4-1. Sign-on Profile Data Base

A description of each segment layout follows.

1. Project/Group Segment (PG)

   Segment Length = 44 bytes

Position	Length	Data Type	Data Description
1	2	alphanum	Project/Group code (DBD sequence field)
3	26	alphanum	Project/Group description for Project Message Sending and Display screens
29	6	num	Reserved
35	4	num	Reserved
39	4	alphanum	Major application system identification
43	2	alphanum	Reserved

2. User Identification (SR) Segment

   Segment Length = 44 bytes

Position	Length	Data Type	Data Description
1	6	alphanum	User identification (DBD sequence field)
7	11	alphanum	User name
18	6	alphanum	Reserved
24	2	alphanum	Profile ID indicating which PR segment(s) contains the user's list of eligible transactions
26	8	alphanum	Reserved (lockword)
34	11	alphanum	Reserved

3. Profile Identification (PR) Segment

   Segment Length = 334 bytes

Position	Length	Data Type	Data Description
1	3	alphanum	Profile ID and continuation (DBD sequence field)
4	3	alphanum	ID of continuation SEG
7	4	alphanum	Reserved
11	2	alphanum	Reserved
13	2	binary	Number of transaction table entries

(The following four bytes are repeated at appropriate offsets for each entry of the table.)

15	2	alphanum	Transaction ID represented by segment or special processing ID
17	1	num	Level of operation allowed for this transaction (transaction mode 1 to 6)
18	1	alphanum	0,1, 2 or blank - show this transaction ID on the secondary option menu. 3 - do not show it, but still allow it to be selected by a user or by a transaction switch.

4. Message (MG) Segment

Segment Length = 156 bytes

Position	Length	Data Type	Data Description
1	22	alphanum	DBD sequence field indicating origin, date, and time of message generation.

(Breakdown of preceding format)

1	4	alphanum	'USER' for Project/Group Message Sending, 'INFO' for a broadcast by a Data Base Administrator, or '0000' for automatic message sending
5	6	alphanum	Date as YYMMDD
11	8	alphanum	The time message is sent (HHMMSSHH)
19	4	alphanum	(PG) Project/Group originating the message if generated by Project/Group Message Sending, or (nnnn) the message number from the Message Data Base if generated by automatic message sending
23	1	alphanum	Acknowledge flag. An asterisk '*' indicates a message has not been acknowledged. A blank ' ' indicates that the message has been acknowledged.
24	127	alphanum	Actual message text

Updating the Sign-on Profile Data Base (DL/I Version)

The Sign-on Profile Data Base must be updated when any one of the following new IDs is added to the conversational environment:

- Major Application System ID
- Project/Group ID
- User ID
- Transaction ID

Standard Processing transactions are provided to add or modify segments in the Sign-on Profile Data Base. These transactions are available to authorized users for online processing.

An authorized user signs on to IMSADF II and selects Option D, Standard Segment processing. The following is a description of the data that must be supplied and the screens that will be displayed when adding Project/Group (PG), User Identification (SR), and Profile Identification (PR) segments:

1. Project/Group (PG) Segment

   On the Primary Option Menu, the user selects:

   - Option 'D' (Transaction Selection)
   - Transaction Mode '2' (Initiate)
   - Segment ID 'PG'
   - Key 'project/group'

   Transaction mode 2 (initiate) must be specified, since the Project/Group segment is the root segment of the Sign-on Profile Data Base. The Segment ID is PG and the key is the two-character code for the Project/Group to be added. Figure 4-2 illustrates the data needed to build the PG segment.

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

INITIATE                    TRANSACTION: PROJECT GROUP
OPTION: _   TRX: 2PG  KEY:   QQ
   *** ENTER DATA FOR ADD ***
        PROJECT/GROUP--- QQ
        DESCRIPTION----- SAMPLE CHECKOUT
        MAJOR SYSTEM ID- SAMP
```

Figure 4-2. Data Needed to Build the PG Segment

2. User Identification (SR) Segment

When the 'Segment Added Successfully' message is received for the PG segment, the user selects from the Primary Option Menu:

- Option 'D' (Transaction Selection)
- Transaction Mode '4' (Add)
- Segment ID 'SR'
- Key 'project/group and six-character employee number'

The next screen displayed is the SR segment screen. Figure 4-3 illustrates data needed to build the SR segment.

The INFO field is an eight-byte field which could be used optionally as a lockword.

```
            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 4-3. Data Needed to Build the SR Segment

3. Profile Identification (PR) Segment

To add a PR segment, the user selects from the Primary Option Menu:

- Option 'D' (Transaction Selection)
- Transaction Mode '4' (Add)
- Segment ID 'PR'
- Key 'two-character project/group and two-character profile ID'

The next screen that will be displayed is the PR segment display screen. Figure 4-4 illustrates the data that must be supplied to build the PR segment. Refer to the layout of the PR Segment shown in "Sign-on Profile Data Base Layout (DL/I Version)" for a description of the transaction profile line entries.

```
                         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--- 5
PROFILE LINE  1- PA10PD30IV30CY30CD52
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 4-4. Data Needed to Build the PR Segment

4. Message (MG) Segment

The Message Segment (MG) is not built directly by a user, but through the Automatic Message Sending or Project/Group Message Sending functions.

The procedures discussed above pertain to adding segments to the Sign-on Profile Data Base. These same procedures can be followed to update an existing segment. The only exception is that the transaction mode would be 5 (update). An example would be that the PR segment needs to be updated when a new transaction ID is added to an application.

These procedures skip the Key Selection screens. The Key Selection screens will be displayed when the user does not enter the key on the Primary Option Menu. Note that transaction mode, Segment ID, and Key data also can be entered from the Segment Display screen, in which case the display of the Primary Option Menu, Secondary Option Menu, and Key Selection screens will be bypassed.

SIGN-ON PROFILE DATA BASE (DB2 VERSION)

The following sections describe the DB2 version of the Sign-on Profile Data Base. If this option is selected, the Sign-on Profile data is contained in four DB2 tables. Each table provides the function contained in one of the segments in the DL/I version.

The Project/Group (S1) table contains the four-character application system identification. In conversational processing, this identification must match the MOD name entered in the /FOR xxxx command. If it does not, the sign-on is invalid. In batch processing, the four-character application system ID in the S1 table must match the first four characters of the batch transaction driver. If not, the sign-on is invalid.

The Profile Identification (S3) table is keyed by Project/Group and a two-character profile ID. It contains a list of transaction selections and authorized modes of operation. A maximum of 300 transactions can be carried in a S3 entry.

Each row in the User Identification (S2) table contains a profile ID field that associates the employee with the appropriate S3 table entry. In Conversational processing, a Secondary Option Menu screen is tailored for each user by comparing data contained in the Secondary Option Menu Rule with that contained in the Profile Identification (S3) table for that user. The purpose of this comparison is to identify which transactions can be selected by the user. Those transactions that appear in both the Secondary Option Menu Rule and in the S3 table are displayed on the Secondary Option Menu screen. In this way, only those selections that are valid for the user are presented.

In batch processing, the transaction specified in the transaction input must appear in the user's S3 table entry. If it doesn't, the input transaction is invalid.

The transaction mode entered by the user is also validated against data in the S3 table. Each transaction listed in the table entry is assigned an authorized mode of operation (1 to 6). The transaction specified is the highest level for which a user is eligible. Transaction mode 1 (DELETE) is the highest level possible. Each level includes all lower levels.

The Sign-on Profile Data Base also provides a message (S4) table where messages destined for a Project/Group are collected. These messages can be initiated by a user through the Project/Group Message Sending function or can be generated automatically as a result of data base segment deletion or modification.

Sign-on Profile Data Base Layouts (DB2 Version)

The Sign-on Profile Data Base is comprised of four DB2 tables. A description of each table's layout follows.

1. Project/Group table (S1)

Position	Length	Data Type	Data Description
1	2	alphanum	Project/Group code (key)
3	26	alphanum	Project/Group description for Project Message Sending and Display screens
29	6	num	Reserved
35	4	num	Reserved
39	4	alphanum	Major application system identification
43	2	alphanum	Reserved

2. User Identification (S2) Table

Position	Length	Data Type	Data Description
1	2	alphanum	Project/Group code (key)
3	6	alphanum	User identification (key)
9	11	alphanum	User name
20	6	alphanum	Reserved
26	2	alphanum	Profile ID indicating which PR segment(s) contains the user's list of eligible transactions
28	8	alphanum	Reserved (lockword)
36	11	alphanum	Reserved

3. Profile Identification (S3) Table

Position	Length	Data Type	Data Description
1	2	alphanum	Project/Group code (key)
3	2	alphanum	Profile ID (key)
5	10	alphanum	Reserved
15	2	binary	Number of trans-action table entries

(The following four bytes are repeated at appropriate offsets for each entry of the table. Maximum of 300 entries).

15	2	alphanum	Transaction ID represented by segment or special processing ID
17	1	num	Level of operation allowed for this transaction (transaction mode 1 to 6)
18	1	alphanum	0,1, 2 or blank - show this transaction ID on the secondary option menu. 3 - do not show it, but still allow it to be selected by a user or by a transaction switch.

4. Message (S4) Table

Position	Length	Data Type	Data Description
1	2	alphanum	Project/Group code (key)
3	22	alphanum	key indicating origin, date, and time of message generation.

(Breakdown of preceding format)

3	4	alphanum	'USER' for Project/Group Message Sending, 'INFO' for a broadcast by a Data Base Administrator, or '0000' for automatic message sending
7	6	alphanum	Date as YYMMDD
13	8	alphanum	The time message is sent (HHMMSSHH)
21	4	alphanum	(S1) Project/Group originating the message if generated by Project/Group Message Sending, or (nnnn) the message number from the Message data base if generated by automatic message sending
25	1	alphanum	Acknowledge flag. An asterisk '*' indicates a message has not been acknowledged. A blank ' ' indicates that the message has been acknowledged.
26	127	alphanum	Actual message text

Updating the Sign-on Profile Data Base (DB2 Version)

The Sign-on Profile Data Base must be updated when any one of the following new IDs is added to the conversational environment:

- Major Application System ID
- Project/Group ID
- User ID
- Transaction ID

Standard Processing transactions are provided to add or modify rows in three of the Sign-on Profile tables. These transactions are available to authorized users for online processing.

An authorized user signs on to IMSADF II and selects Option D, Standard Segment processing. The flow of screens will look like those shown in "Updating the Sign-on Profile Data Base (DL/I Version)" on page 4-5. Some screens will be slightly different. The following transaction IDs should be selected to perform the updates.

Transaction ID	Transaction Process
S1	Maintains the Project/Group table
S2	Maintains the User Identification table
S3	Maintains the Profile Identification table
S4	Maintains the Message table

Note: The Message table (S4) is not built directly by a user, but through the Automatic Message Sending or Project/Group Message Sending functions.

MESSAGE DATA BASE

The Message Data Base contains data that enables the system to perform the following functions, each of which is discussed in detail below.

1. Determine automatic message routing to Project/Groups or individual users

2. Generate messages for automatic message sending or error/warning message response

3. Determine routing of secondary transactions or alternate output messages

4. Collect messages destined for an individual user

5. Screen and message HELP facility.

MESSAGE DATA BASE (DL/I VERSION)

The following section describes the processing of the five Message data base functions under a DL/I data base environment.

Automatic Message Routing

The routing of automatic messages is determined by retrieving the appropriate Header (AH) and System (AR) segments from the Message Data Base.

The key of the root segment (AH) has four formats, which allow specification of routing information for specific transaction processing or global routing.

Format 1 is used for all unconditional message routing, and may also be used for conditional message routing.

The format (spgmxxnn) is developed from the first character of the system ID currently being processed, the project/group code, the transaction mode, the two-character segment or Special Processing id, and a message group code. With this format, unconditional message routing occurs when a segment is added or deleted, and the Rules Generator operand DAMSG=YES is specified. The message sending module builds the key from the current processing information, and a message group code of 00. If a routing segment is not found, processing continues.

Conditional message routing is triggered by the Message Sending leg of the Audit Data Base. If a field requires message sending, the Auditor attaches a four-digit code to the field Segment Layout entry. This code (0Cnn) specifies the key format of the (AH) segment, and the message group code (01-99) to use. The format is determined by the C value (0-3).

If	C=0	then the key format is	spgmxxnn
	C=1		ss######
	C=2		ss##pg##
	C=3		ss####nn
			(#s are constant)

By using this technique, routing segments can apply to a specific transaction process, an entire application system, or a project/group or message group code within an application system.

When an AH segment is found, the dependent AR segments are retrieved to determine which Project/Groups and individual users are to receive the message and which message numbers are to be used to construct the message.

Message Generation

Message generation employs the Message Data Base to build a full text message using the four-character application system ID and a four-digit message number. The message number is obtained from either Automatic Message Sending or Audit error processing.

A Header (HD) segment in the format ssssnnnn is retrieved using the four-character system ID currently in process and a message number. Information in the HD segment includes the message length and up to 5 mapping requirements. The mapping requirements enable the message text to include segment and variable data pertinent to the transaction currently in process. The message text is contained in the dependent SY segments. If mapping requirements exist for a message, the variable data is mapped into the actual message text and the message is ready to be sent to its destination, which will be either a Project/Group, a user, or a terminal.

Secondary Transactions

The destinations for secondary transactions and output message routing are determined by the appropriate SD and LT segments. When an STX operand for an Input Transaction Rule indicates that a secondary transaction or alternate output message is to be generated, the message content is defined by the named Output Format Rule and message routing information is obtained from the Message Data Base. The message can be routed to the entering terminal, to an alternate terminal, or to a program. In addition, message routing may be dependent on the entering terminal.

The SD segment consists of a DBD sequence field equal to the Output Format Rule name and a default destination field equal to a terminal name, transaction name, or IOPCB. ('IOPCB' specifies the entering terminal.)

If message routing is to depend on the entering terminal source, an LT segment is retrieved using the name of the entering terminal as the DBD sequence field. The LT segment is then used to route the message to up to eight specified terminals. If an LT segment does not exist, then the default destination in the HD segment is used for routing the message.

User Messages

Messages destined for individual users are maintained in the Message Data Base. The UH segment provides a User Header, which is an employee's USERID. The dependent User Message (US) segments contain the message text. User messages originate from the User Message Sending or the Automatic Message Sending functions and are displayed by the User Message Display function.

HELP Facility

A HELP facility is available in conversational processing, to describe the purpose and input requirements for the currently displayed screen and to expand error message text on the transaction error screen. HELP 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)

HELP is invoked through the entry of '?'

- 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 Error screens and
- in the SELECT field of the Secondary Option Menu screen.

Screen related HELP text is accessed under the appropriate HE and HT segments. The HE key is constructed to uniquely relate HELP data to a conversational screen or set of screens.

HE KEY FORMAT	ASSOCIATED SCREEN
ssssESO@ or ????ESO@	Sign-on 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 screen on which the '?' is entered will determine the key format used to access the HELP text. If the HE segment is not present the message 'HELP INFORMATION FOR THIS SCREEN IS NOT AVAILABLE' is displayed.

A HELP request for error messages is invoked from either the screen requesting the display of errors with 'ENTER E TO DISPLAY ERROR OR WARNING MESSAGE' displayed or the error screen currently displaying errors. This HELP text is contained in one or more MH segments which are children of the last SY segment in the error message. Requesting HELP for error messages causes HELP text to be displayed for all messages. Error messages which do not have HELP text provided will display the message 'HELP INFORMATION FOR THIS MESSAGE IS NOT AVAILABLE'.

Message Data Base Layout (DL/I Version)

The Message Data Base is a three-level HDAM (OSAM) data base. Figure 4-5 illustrates the hierarchical structure of this data base. It shows the segment names known to IMS/VS.

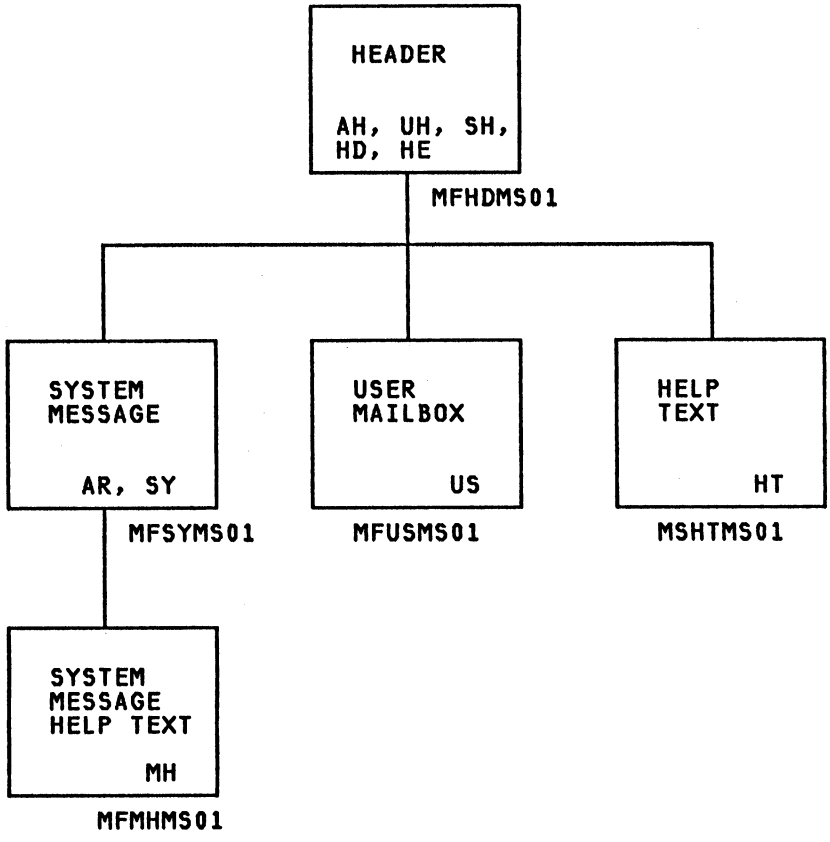


Figure 4-5. Message Data Base

Since Message Data Base segments are multipurpose, the segment format will be described for each usage. Except segment US, each segment in the Message Data Base is 78 bytes in length and has an eight-character DBD sequence field starting in position 1. The US segment is 156 bytes andd has a 22-byte DBD sequence field.

1. Automatic Message Routing

Automatic Message Routing Header Segment (AH) (Alias for HD)

Position	Length	Data Type	Data Description
1	8	alphanum	DBD sequence field developed from the system ID, transaction name, processing mode, and automatic message sending requirements.

(Breakdown of above format)

Format 1 (Format code = 0) spgmxxnn

1	1	alphanum	First character of major system identification.
2	2	alphanum	Project/Group. For nonconversational and batch processing without Sign-on, 2 and 3 contain '00'.
4	1	alphanum	Transaction mode of operation (1 through 6).
5	2	alphanum	Two-character transaction or Special Processing ID from IMSADF II transaction.
7	2	alphanum	Message group code (01 to 99) from message leg of the Audit Data Base or 00 for unconditional message generation.

Format 2 (Format code = 1) ss######

1	2	alphanum	Two-character major system identification.
3	6	alphanum	###### (constant)

Format 3 (Format code = 2) ss##pg##

1	2	alphanum	Two-character major system identification.
3	2	alphanum	## (constant)
5	2	alphanum	Project/Group - For nonconversational and batch processing, 00
7	2	alphanum	## (constant)

Format 4 (Format code = 3) ss####NN

1	2	alphanum	Two-character major system identification.
3	4	alphanum	#### (constant)
7	2	alphanum	Message group code (01-99)
9	70	blank	Not used

Automatic Message Routing Address Segment (AR) (Alias for SY)

Position	Length	Data Type	Data Description
1	8	numeric	DBD sequence field. An automatic message routing SY segment exists for each Project/Group or user receiving a message.
9	2	alphanum	Project/Group to receive the message(s). Messages are placed in the Project/Group's message (MG) segment of the Sign-on Profile data base. The Project/Group specification is optional if a user address is specified.
11	4	numeric	Message number of the message to be generated by the Message Generator. A maximum of 5 messages can be specified.
15	4	numeric	Message number
19	4	numeric	Message number
23	4	numeric	Message number
27	4	numeric	Message number
31	8	alphanum	User address in the Message data base at which messages are received. Messages are placed in the User Message (US) segments under the appropriate User Header (HD) segment. The user address is optional.

2. Message Generation

Header Segment (HD)

Position	Length	Data Type	Data Description
1	8	alphanum	DBD sequence field developed from major system ID and message number.

(Breakdown of above format)

1	4	alphanum	Major application system identification.
5	4	alphanum	Unique message number. It is recommended that user message numbers range between 2000 to 9999 to prevent conflicts with the IMSADF II system messages on the error display or the batch transaction register.
9	4	numeric	Total message length. A maximum of 127 bytes is allowed for a message to a Project Group or user address and of 980 bytes for a message to an error screen. The message can be spread across 14 SY segments. The eight-byte key of the SY segment is not included in the length calculation.

A maximum of five data mappings are allowed for the message text. The field name in a Segment Layout Rule, VARLIST index, or keyword indicates where the segment or variable data is located. The offset indicates where the data is to be moved in the message text. The field name must be the field name in a Segment Layout Rule that is used in the transaction currently in process. VARLIST provides a technique for mapping data other than segment data into the message text. By specifying VARLIST1 for Segment Handler Rule error messages, the two-character DL/I status code will be mapped. If VARLIST2 is specified in an audit error message, the three character transaction mode and segment ID will be mapped. If VARLIST3 is specified in an audit or Segment Handler error message, the six-character userid will be mapped into the message text at the specified offset. If VARLIST4 is specified in an audit error message, the Audited field name (eight characters) will be mapped into the message text at the specified offset. If VARLIST5 is specified in an audit error message, the Audited field value will be mapped into the message text at the specified offset.

If VARLIST6 is specified in an audit error message the DB2 SQLCODE (five characters) will be mapped into the message. If VARLIST7 is specified in an audit error message the eight DB2 warning characters (W) will be mapped into the message at the specified offset.

Specific fields from the SPA can be inserted into the message by specifying one of the keywords defined under FIELD statement, KWNAME=operand.

Position	Length	Data Type	Data Description
13	8	alphanum	Field name in Segment Layout Rule or VARLISTn to be mapped 'from'.
21	3	numeric	Offset in message text where the data is mapped 'to'. Offset 1 is equal to position 9 in the segment, since the DBD sequence field is not part of the message text.
24	1	blank	Reserved

(Positions 25 to 73 repeat the above 3 formats for mapping to the message text.)

System Message Segment (SY)

Position	Length	Data Type	Data Description
1	8	numeric	DBD sequence field used for Text Utility Processing to modify the SY segment. There may be more than one SY segment containing message text for each HD segment used by message generation.
9	70	alphanum	Message text for message generation. If variable data is to be mapped into the message text, sufficient blanks or other filler data must be left to accommodate the data being mapped.

3. Secondary Transaction/Output Message Routing

Secondary Transaction Destination Segment (SD) (Alias for HD)

Position	Length	Data Type	Data Description
1	8	alphanum	DBD sequence field and the name of the OFR specified in an Input Transaction Rule for generating a secondary transaction or output message.
9	8	alphanum	Default destination for output message or secondary transaction.
17	62	alphanum	Comments for user information.

Logical Terminal Segment (LT) (Alias for SY)

Position	Length	Data Type	Data Description
1	8	alphanum	DBD sequence field identifying and entering terminal name.

The following one to eight names specify the terminals to which the output message is to be routed when the entering terminal is the terminal identified in positions 1 to 8. A value of IOPCB indicates that the message should be routed to the entering terminal.

9	8	alphanum	Alternate terminal name.

Positions 17 to 73 contain alternate terminal names that are up to eight characters in length.

4. User Message

User Header Segment (UH) (Alias for HD)

Position	Length	Data Type	Data Description
1	8	alphanum	DBD sequence field defining the User Header for the collection of user messages in the US segment. The user header consists of the user's six-character USERID padded with blanks.
9	8	alphanum	Reserved
17	22	alpha	Employee name
31	69		Unused

User Message Segment (US) (segment length = 156 bytes)

Position	Length	Data Type	Data Description
1	22	alphanum	DBD sequence field indicating origin, date, and time of message generation.

(Breakdown of preceding format)

1	4	alphanum	'USER' for Project/Group Message Sending or '0000' for automatic message sending.
5	6	alphanum	Date as YYMMDD.
11	8	alphanum	The time message is sent (HHMMSSHH).
19	4	alphanum	(PG) Project/Group originating the message if

			generated by Project/Group Message Sending, or (nnnn) the message number from the Message data base if generated by automatic message sending.
23	1	alphanum	Acknowledge character for User Message Display. '*' = unacknowledged message ' ' = acknowledged message
24	127	alphanum	Message Text from user Message or Automatic Message Sending.

5. HELP Facility

HELP Header (HE) (Alias for HD)

Position	Length	Data Type	Data Description
1	8	alphanum	Segment sequence field (key)
9	70		reserved

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.

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 followin; 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

If a 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 (DEFADF). The default is '$$'. This causes an insert of only the necessary data.

Note that the HEADER only needs to be specified in the first HT segment.

Updating the Message Data Base (DL/I Version)

The Message Data Base must be updated when Automatic Message Sending requirements are changed, when secondary transaction or output message routing requirements are changed, when additional information or error messages are generated, or when a new user is added. Standard and Text Utility transactions are provided to add or modify segments in the Message Data Base. These transactions are available to authorized Facility users for online processing.

Since the same Message Data Base segments are used for different functions, the transactions for building and modifying segments will be described by function. The following description defines the data that must be supplied and the screens that will be displayed. These descriptions are concerned primarily with the Initiate and Add transaction modes. However, the other processing modes (Delete, Remove, Update, and Retrieve) also can be performed by these transactions. Note that in the following discussion, only the segment display screens are shown. All Transaction Mode, Segment ID, and Key data was entered on the Primary Option Menu and Key Selection screens, which are not shown here. The user can obtain the Key Selection screens by not entering key data.

Automatic Message Routing

An automatic Message Routing Header (AH) segment and Automatic Message Routing Address (AR) segment must be built to coordinate a data base segment modification with appropriate message text and routing requirements. The Standard transactions AH and AR are provided to build the AH and AR segments in the Message Data Base.

An authorized user signs on and selects:

- Option 'D' (Segment Selection)
- Transaction Mode '2' (Initiate)
- Segment ID 'AH'
- Key 'spgmxxn'

In the key format, spgmxxnn

s = First character of major application system ID

pg = Project/Group or 00 for nonconversational or batch processing

m = Transaction mode (1 to 6) for which message is to be generated.

xx = Two-character Standard or Special Processing ID from the transaction

nn = Message group code (01 to 99) from the message leg of the Audit Data Base or 00 for unconditional message generation

Entry of this Primary Option Menu screen results in the display of the AH segment screen illustrated in Figure 4-6.

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

ADD                          TRANSACTION: MESSAGE ROUTING HEADER
OPTION:     TRX: 4AH  KEY:   SZZ1PA00
   *** ENTER DATA FOR ADD ***
        SYSTEM ID---------- S
        PROJECT/GROUP------ ZZ
        MODE--------------- 1
        SEGMENT ID--------- PA
        MESSAGE GROUP CODE- 00
```

Figure 4-6. Data Needed to Build the Automatic Message Routing Segment (AH)

The AH segment is added to the Message Data Base by entering this screen to the system. When the SEGMENT ADDED SUCCESSFULLY message is received, the user has added the AH segment to the Message Data Base.

To add an AR address segment, the user selects from the Primary Option Menu:

- Option 'D' (Segment Selection)
- Transaction Mode '4' (Add)
- Segment ID 'AR'
- Key 'spgmxxnn00000001'

Figure 4-7 shows the AR segment Display screen that results from this entry.

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

ADD                          TRANSACTION: MESSAGE ADDRESS SEGMENT
OPTION:     TRX: 4AR  KEY:    SZZ1PA0000000001
   *** ENTER DATA FOR ADD ***
        SYSTEM ID---------- S
        PROJECT/GROUP------ ZZ
        MODE--------------- 1
        SEGMENT ID--------- PA
        MESSAGE GROUP CODE- 00
        SEQUENCE NUMBER---- 00000001
        PROJECT/GROUP------ __
        MESSAGE NUMBER 1--- 0000
        MESSAGE NUMBER 2--- 0000
        MESSAGE NUMBER 3--- 0000
        MESSAGE NUMBER 4--- 0000
        MESSAGE NUMBER 5--- 0000
        USER HEADER-------- _______
```

Figure 4-7. Address Segment (AR) Screen Before Entry

On the segment Display screen for AR in Figure 4-8, the user enters the Project/Group and/or user to whom the message is to be routed and the 1 to 5 message numbers that will be used to generate the messages. The Project/Group entered is the two-character Project/Group code from the Sign-on Profile Data Base. The user address is the six-character USERID in the User Header segment. The message numbers are four-digit numbers from the Message Data Base.

When the SEGMENT ADDED SUCCESSFULLY message is received, the automatic message routing AR segment has been added to the Message Data Base. An AR segment must be added for each Project/Group and each user who will require the message, because the AR segment maintains only one Project/Group and one user address.

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

ADD                          TRANSACTION: MESSAGE ADDRESS SEGMENT
OPTION:    TRX: 4AR  KEY:   SZZ1PA0000000001
   *** ENTER DATA FOR ADD ***
        SYSTEM ID---------- S
        PROJECT/GROUP------ ZZ
        MODE--------------- 1
        SEGMENT ID--------- PA
        MESSAGE GROUP CODE- 00
        SEQUENCE NUMBER---- 00000001
        PROJECT/GROUP------ MX
        MESSAGE NUMBER 1--- 1000
        MESSAGE NUMBER 2--- 0000
        MESSAGE NUMBER 3--- 0000
        MESSAGE NUMBER 4--- 0000
        MESSAGE NUMBER 5--- 0000
        USER HEADER-------- _______
```

Figure 4-8. Data Needed to Build the Automatic Message Address Segment (AR)

Message Generation

A message built for automatic message sending or error/warning message message generation is constructed from a Header (HD) segment and dependent System Message (SY) segments. The Header segment contains the message number and any message text mapping requirements. The System Message segments contain the message text. The Standard transaction HD, and the Text Utility transaction SY, are provided to build the HD and SY segments.

An authorized user signs on and selects from the Primary Option Menu:

- Option 'D' (Segment Selection)
- Transaction Mode '2' (Initiate)
- Segment ID 'HD'
- Key 'ssssnnnn'

In the key format ssssnnnn:

ssss = major application system ID

nnnn = Message number (unique to this application system ID)

The HD segment screen is displayed and the message length and any mapping requirements must be specified thereon by the user. The MESSAGE LENGTH field is provided for entering the entire message text length. The FIELD NAME and MESSAGE OFFSET fields are provided for entering segment or variable mapping definitions. Note that message length only pertains to message text and does not include the DBD sequence field. The FIELD NAME field specifies the field name of a Segment Layout Rule or VARLISTn. Specification of a Segment Layout Rule field name causes data to be mapped from a data base segment. Specification of VARLISTn causes data to be mapped from some other data location. VARLIST1 maps the DL/I status code, VARLIST2 maps the transaction mode and the Processing ID (Standard or Special) currently in process, VARLIST3 maps the USERID currently signed on, VARLIST4 maps the audited field name, and VARLIST5 maps the audited field value into the message text. The MESSAGE OFFSET field specifies the position in the message text where data mapping is to start. Note that the first offset is the beginning of message text and the eight-character DBD sequence field is ignored in the system message text (SY) segment. Figure 4-9 illustrates the screen used to build an HD segment.

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

ADD                          TRANSACTION:  MESSAGE GENERATION HEADER
OPTION:     TRX: 4HD  KEY:   SAMP9990
   *** ENTER DATA FOR ADD ***
        MESSAGE NUMBER ------- SAMP9990
        MESSAGE LENGTH ------- 0070
        FIELD NAME 1 --------- SACDDIPU
        MESSAGE OFFSET 1 ----- 014
        FIELD NAME 2 --------- VARLIST2
        MESSAGE OFFSET 2 ----- 055
        FIELD NAME 3 --------- ________
        MESSAGE OFFSET 3 ----- 000
        FIELD NAME 4 --------- ________
        MESSAGE OFFSET 4 ----- 000
        FIELD NAME 5 --------- ________
        MESSAGE OFFSET 5 ----- 000
```

Figure 4-9. Data Needed to Build the Message Generation Header Segment (HD)

```
                    M E S S A G E   D A T A   B A S E
UPDATE                       TRANSACTION: SYSTEM MESSAGE TEXT
          TRX: 5SY  KEY:   SAMP9990

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
```

Figure 4-10. Text Utility Display for SY Segment.

When the SEGMENT ADDED SUCCESSFULLY message is received, the user has added the HD segment to the Message Data Base. To add SY text segments, the user selects on the Primary Option Menu:

- Option 'D' (Segment Selection)
- Transaction Mode '5' (Update)
- Segment ID 'SY'
- Key 'ssssnnnn'

If no SY text segments exist, the Text Utility screen will be displayed as shown in Figure 4-10. The DBD sequence numbers and message text may now be entered. Sufficient space must be left for any data mapping that

is to be performed. In the example illustrated in Figure 4-11, one character is allowed for segment field SACDDIPU data mapping and two characters are provided for the VARLIST2 mapping of the processing mode and segment ID. If additional lines of text are necessary, additional SY segments are added using Text Utility Processing. For each additional line of message text, the user adds an eight-character sequence field followed by the message text.

```
                    M E S S A G E   D A T A   B A S E
UPDATE                     TRANSACTION: SYSTEM MESSAGE TEXT
          TRX: 5SY  KEY:   SAMP9990

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
00000001 DISBURSEMENT CODE INCORRECT (-) SPECIFY P OR U. TRXID = --.
```

Figure 4-11. Text Utility for SY Segment Text

Secondary Transaction/Output Message Routing

The Message Data Base segments, HD and SY, provide routing control for secondary transactions and output messages. The Standard Processing transactions, SD and LT, provide online update capability for the Message Data Base. The SD transaction builds or updates the HD segment, and the LT transaction builds or updates the SY segment.

An authorized user signs on and selects from the Primary Option Menu:

- Option 'D' (Segment Selection)
- Transaction Mode '2' (Initiate)
- Segment ID 'SD'
- Key 'output format rule name'

The Output Format Rule naming convention is ssORxx01, where ss is the first two characters of the system identification, OR is a constant, xx is the Output Format Rule ID, and 01 is a constant. The SD segment screen is displayed, and the DEFAULT ALTERNATE IOPCB and COMMENTS fields are supplied for input as illustrated in Figure 4-12. DEFAULT ALTERNATE IOPCB is the destination for the message. The destination may be:

- an IMS/VS transaction name
- a CICS Transient Data queue name
- a terminal name
- 'IOPCB'

IOPCB indicates that the message is to be routed to the entering terminal.

Comments of up to 62 characters also can be specified for user information.

```
                              MESSAGE DATA BASE
 ADD          DATABASE: MESSAGE                SEGMENT: HEADER
OPTION:      TRX: 4SD KEY: SAORIE01
 *** 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 ------ SAORIE01
DEFAULT ALTERNATE PCB ----- LTERM3
COMMENTS --- ADD ERROR NOTIFICATION FOR SEGMENT IV.
```

Figure 4-12. Data Needed to Build the Secondary Transaction Destination Segment (SD)

When the SEGMENT ADDED SUCCESSFULLY message is received, the LT segments can be added. From the Primary Option Menu, the user selects:

- Option 'D' (Segment Selection)
- Transaction Mode '4' (Add)
- Segment ID 'LT'
- Key 'output format rule name - source terminal name'

The LT segment screen is displayed and provides up to eight ALTERNATE TERMINAL NAME fields for entry. The ALTERNATE TERMINAL names specify the terminal destinations when the source terminal is the entering terminal. IOPCB indicates the entering terminal. In the example shown in Figure 4-13, the output message defined by the SAORIE01 Output Format Rule will be sent to both LTERM1 and to LTERM3 when the transaction currently in process is entered from L3277099. There must be an LT (SY) segment for each source terminal that requires qualified output message routing. Otherwise, the default destination is used from the SD segment.

```
                              MESSAGE DATA BASE
 ADD          DATABASE: MESSAGE                SEGMENT: LOGICAL TERMINAL
OPTION:      TRX: 4LT KEY: SAORIE01L3277099
 *** 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 -------- SAORIE01
INPUT LOGICAL TERMINAL NAME - L3277099
ALTERNATE TERMINAL 1 NAME --- LTERM1
ALTERNATE TERMINAL 2 NAME --- LTERM3
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 4-13. Data Needed to Build the Logical Terminal Segment (LT)

User Messages

The Message Data Base provides a collection facility for user messages. To implement the user message collection facility, a user header must be established in the HD segment to identify an employee uniquely. User messages then can be stored in US segments under the proper user message header (UH) segment.

To build a user header, the Standard Processing transaction, UH, is provided. An authorized user signs on and, from the Primary Option Menu, selects:

- Option 'D' (Segment Selection)
- Transaction Mode '2' (Initiate)
- Segment ID 'UH'
- Key 'employee userid'

The UH screen is displayed with the Employee Name field available for optional input. Refer to the screen illustrated in Figure 4-14. Once the User Header (UH) segment has been added to the Message Data Base, user messages, initiated through either User Message Sending or Automatic Message Sending, can be sent to the defined collection point.

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

ADD                          TRANSACTION: USER MESSAGE HEADER
OPTION:     TRX: 4UH  KEY:    999999
   *** ENTER DATA FOR ADD ***
        USER ID- 999999
        NAME---- J. SMITH
```

Figure 4-14. Data Needed to Build the User Header Segment (UH)

The US segments are built through the User Message Sending or the Automatic Sending functions and require no user preparation other than the creation of the User Header (UH) root segment.

HELP Facility

The Message Data Base segments HE, HT and MH are used to control and store the HELP text.

The HE segment is the header for the Screen HELP facility. To add an HE segment to the Message Data Base, the user selects from the Primary Option Menu:

- Option 'D' (Segment Selection)
- Transaction Mode '4' (Add)
- Segment ID 'HE'
- Key (as required by screen for which header is created)

The HE screen displays only the key and upon entry the segment is added to the Message Data Base. An alternative is to add both the HE and HT segments on the HT screen, since the HT transaction has insert eligibility for HE. The HE screen in Figure 4-15 demonstrates how to build a HELP header for a Sign-on screen for application system SAMP.

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

ADD                          TRANSACTION: HELP HEADER
OPTION:    TRX: 4HE  KEY:   SAMPESO@
   *** ENTER DATA FOR ADD ***
        HELP HEADER KEY- SAMPESO@
```

Figure 4-15. HE-HELP Header Generation Screen

The HT segment contains the screen related HELP text. HT segments will be displayed exactly as the 79 byte lines of help text are built. To add a HT segment to the Message Data Base, the user selects from the Primary Option Menu:

- Option 'D' (Segment Selection)
- Transaction Mode '4' (Add)
- Segment ID 'HT'
- Key 'as required by screen for which text is generated'

The HT screens in Figure 4-16 and Figure 4-17 demonstrate 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 4-16. HT-HELP Generation Screen (Page 1 of 2)

```
                        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 4-17. HT-HELP Generation Screen (Page 2 of 2)

The MH segments contain the HELP text for error messages. They are child segments of the last SY segment in an error message. To add an MH segment to the Message Data Base, the user selects from the Primary Option Menu:

- Option 'D' (Segment Selection)
- Transaction Mode '4' (Add)
- Segment ID 'MH'
- Key 'ssssnnnn00000010001'

The MH screen in Figure 4-18 demonstrates input to build message HELP text for the SAMP9999 error message.

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

ADD                                  TRANSACTION: MESSAGE HELP TEXT
OPTION:     TRX: 4MH   KEY: SAMP9999000000010001
*** ENTER DATA FOR ADD ***
MESSAGE NUMBER  (SSSSNNNN): SAMP9999
MESSAGE LENGTH  (SY)------: 00000001
                      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 ( ) 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 4-18. MH-Message Help Generation Screen

If a HT or MH segment is not filled, the end of message characters should be specified after the last valid character. End of message characters are defined during installation and default to $$.

MESSAGE DATA BASE (DB2 VERSION)

The following section describes the processing of the Message data base functions under a DB2 data base environment. Under this environment the Message Data Base is comprised of seven DB2 tables. The tables hold the following information:

Table ID	Table content
M1	Message generation information
M2	Message text
M3	Help text for error messages
M4	Message routing information
M5	Help text for screens
M6	Secondary transaction routing
M7	User mailbox

Automatic Message Routing

The routing of automatic messages is determined by retrieving the appropriate row from the M4 table. The key of the routing information has the same format as described in "Automatic Message Routing" on page 4-12.

The retrieved row contains the project/group and/ or userids which receive the message and the message numbers used to construct the message. Multiple rows can exist, if messages are sent to more than one project/group or userid.

Message Generation

Message generation employs the M1 table to build a full text message using the four-character application system ID and a four-digit message number. The message number is obtained from either Automatic Message Sending or Audit error processing.

The key of the M1 table is ssssnnnn where ssss is the four-character system ID and nnnn is a message number. Information in this table includes the message length and up to 5 mapping requirements. The mapping requirements enable the message text to include segment and variable data pertinent to the transaction currently in process. The message text is contained in the associated M2 table. If mapping requirements exist for a message, the variable data is mapped into the actual message text and the message is ready to be sent to its destination, which will be either a Project/Group, a user, or a terminal.

Secondary Transactions

Secondary transaction and output message routing is determined by information in the M6 table. When an STX operand for an Input Transaction Rule indicates that a secondary transaction or alternate output message is to be generated, the message content is defined by the named Output Format Rule and message routing information is obtained from the M6 table. The message can be routed to the entering terminal, to an alternate terminal, or to a program. In addition, message routing may be dependent on the entering terminal.

The M6 table is keyed on the Output Format rule name and optionally the entering terminal name. If the entering terminal key field is blank, the default destination is used. (The 'IOPCB' specifies the entering terminal.)

If message routing depends on the entering terminal source, the key must be present in the record. If a row exists with both the Output Format

name and terminal name, the routing information in it is used. If a row with Output Format name but no terminal name is found, the default destination is used for routing the message.

User Messages

Messages destined for individual users are maintained in the M7 table. User messages originate from the User Message Sending or the Automatic Message Sending functions and are displayed by the User Message Display function.

HELP Facility

A HELP facility is available in conversational processing, to describe the purpose and input requirements for the currently displayed screen and to expand error message text on the transaction error screen. This facility is maintained in the M3 and M5 tables. The M3 table is used for Help messages associated with error messages and the M5 table is used for Help text associated with screens. The processing of this function is the same as described in "HELP Facility" on page 4-13.

Message Data Base Layout (DB2 Version)

The Message Data Base is comprised of seven DB2 tables.

Each table contains control information for a different dynamic message rule function.

1. Automatic Message Routing

Automatic Message Routing maintained in M4 table

Position	Length	Data Type	Data Description
1	8	alphanum	Key field developed from the system ID, transaction name, processing mode, and automatic message sending requirements.

(Breakdown of above format)

Format 1 (Format code = 0) spgmxxnn

1	1	alphanum	First character of major system identification.
2	2	alphanum	Project/Group. For nonconversational and batch processing without Sign-on, 2 and 3 contain '00'.
4	1	alphanum	Transaction mode of operation (1 through 6).
5	2	alphanum	Two-character transaction or Special Processing ID from ADF transaction.
7	2	alphanum	Message group code (01 to 99) from message leg of the Audit Data Base or 00 for unconditional message generation.

Format 2 (Format code = 1) ss######

1	2	alphanum	Two-character major system identification.

3	6	alphanum	###### (constant)

Format 3 (Format code = 2) ss##pg##

1	2	alphanum	Two-character major system identification.
3	2	alphanum	## (constant)
5	2	alphanum	Project/Group - For nonconversational and batch processing, 00
7	2	alphanum	## constant

Format 4 (Format code = 3) ss####NN

1	2	alphanum	Two-character major system identification.
3	4	alphanum	#### (constant)
7	2	alphanum	Message group code (01-99)
9	8	numeric	Sequential key field 00000001-00000002.. etc.....
17	2	alphanum	Project/Group to receive the message(s). Messages are placed in the Project/Group's message table (S4). The Project/Group specification is optional if a user address is specified.
19	4	numeric	Message number of the message to be generated by the Message Generator. A maximum of 5 messages can be specified.
23	4	numeric	Message number
27	4	numeric	Message number
31	4	numeric	Message number
27	4	numeric	Message number
31	8	alphanum	User key in the M7 table. Messages are placed in the User Message table (M7). The user address is optional.

2. Message Generation

Message Generation maintained in the M1 table

Position	Length	Data Type	Data Description
1	8	alphanum	Key field developed from major system ID and message number.

(Breakdown of above format)

1	4	alphanum	Major application system identification.
5	4	alphanum	Unique message number. It is recommended that user message numbers range between 2000 to 9999 to prevent conflicts with the IMSADF II system messages on the error display or the batch transaction register.

9	4	numeric	Total message length. A maximum of 127 bytes is allowed for a message to a Project Group or user address and of 980 bytes for a message to an error screen. The message can be spread across 14 M2 table rows.
13	8	alphanum	Field name in Segment Layout Rule or VARLISTn to be mapped 'from'.
21	3	numeric	Offset in message text where the data is mapped 'to'. Offset 1 is equal to position 9 in the segment, since the DBD sequence field is not part of the message text.
24	1	blank	Reserved

(Positions 25 to 73 repeat the above 3 formats for mapping to the message text.)

The mapping keywords are explained under "Header Segment (HD)" in the DL/I version of this data base.

3. Message Text

Message Text Maintained in the M2 table

Position	Length	Data Type	Data Description
1	8	alphanum	Key field developed from major system ID and message number.
9	8	numeric	Key sequence field. (00000001,00000002...etc..) There may be more than one M2 row containing message text for each M1 row used by message generation.
17	70	alphanum	Message text for message generation. If variable data is to be mapped into the message text, sufficient blanks or other filler data must be left to accommodate the data being mapped.

4. Secondary Transaction/Output Message Routing

Secondary Transaction Routing maintained in M6 table

Position	Length	Data Type	Data Description
1	8	alphanum	Key field and the name of the OFR specified in an Input Transaction Rule for generating a secondary transaction or output message.
9	8	alphanum	Default destination for output message or secondary transaction.
17	62	alphanum	Comments for user information.
79	8	alphanum	Key field identifying and entering terminal name or blank.

The following one to eight names specify the terminal names to which the output message is to be routed when the entering terminal is the

terminal identified in positions 79 to 86. A value of IOPCB indicates that the message should be routed to the entering terminal.

87	8	alphanum	Alternate terminal name.

Positions 95 to 151 contain alternate terminal names that are up to eight characters in length.

5. User Messages

User Messages maintained in M7 table

Position	Length	Data Type	Data Description
1	8	alphanum	Key field defining the User Header for the collection of user messages. The user header consists of the user's six-character USERID padded with blanks.
9	8	alphanum	Reserved
17	22	alpha	Employee name
39	22	alphanum	Key field indicating origin, date, and time of message generation.

(Breakdown of preceding format)

39	4	alphanum	'USER' for Project/Group Message Sending or '0000' for automatic message sending.
43	6	alphanum	Date as YYMMDD.
49	8	alphanum	The time message is sent (HHMMSSHH).
57	4	alphanum	Project/Group originating the message if generated by Project/Group Message Sending, or (nnnn) the message number from the Message Data Base if generated by automatic message sending.
61	1	alphanum	Acknowledge character for User Message Display. '*' = unacknowledged message ' ' = acknowledged message
62	127	alphanum	Message Text from user Message or Automatic Message Sending.

6. HELP Facility

Message Help maintained in M3 table

Position	Length	Data Type	Data Description
1	8	alphanum	Key (sysid - message number)
9	4	alphanum	Segment sequence field (key)
13	9		reserved
22	79	alphanum	Text that represents one line on the error/warning screen.
101	79	alphanum	Text that represents one line on the error/warning screen.
180	79	alphanum	Text that represents one line on the error/warning screen.

Screen HELP maintained in the M5 table

Position	Length	Data Type	Data Description
1	8	alphanum	Key See "HELP Facility" on page 4-13
9	4	alphanum	sequence key
13	60	alphanum	Header for line 1 of each page.

The following represents 20 lines on the HELP screen.

73	79	alphanum	Text that represents line 5
152	79	alphanum	Text that represents line 6
231	79	alphanum	Text that represents line 7
339	79	alphanum	Text that represents line 8
389	79	alphanum	Text that represents line 9
468	79	alphanum	Text that represents line 10
547	79	alphanum	Text that represents line 11
626	79	alphanum	Text that represents line 12
705	79	alphanum	Text that represents line 13
784	79	alphanum	Text that represents line 14
863	79	alphanum	Text that represents line 15
942	79	alphanum	Text that represents line 16
1021	79	alphanum	Text that represents line 17
1100	79	alphanum	Text that represents line 18
1179	79	alphanum	Text that represents line 19
1258	79	alphanum	Text that represents line 20
1337	79	alphanum	Text that represents line 21
1416	79	alphanum	Text that represents line 22
1495	79	alphanum	Text that represents line 23
1574	79	alphanum	Text that represents line 24

If the row is not filled, the end of message characters should be specified after the last valid character. End of message characters are defined in installation (DEFADF). Default is '$$'. This causes an insert of only the necessary data.

Note that the HEADER only needs to be specified in the first row.

If the user-defined Help message text requires multiple physical pages, then the user must press the PA1 key to retrieve each subsequent page.

Updating the Message Data Base (DB2 Version)

The Message Data Base DB2 tables must be updated when Automatic Message Sending requirements are changed, when secondary transaction or output message routing requirements are changed, when additional information or error messages are generated, or when a new user is added. Standard transactions are provided to add or modify rows in the Message Data Base tables. These transactions are available to authorized Facility users for online processing.

An authorized user signs on to IMSADF II and selects Option D, Standard Segment processing. The flow of screens will look like those shown in "Updating the Message Data Base (DL/I Version)" on page 4-21. Some screens will be slightly different. The following transaction IDs should be selected to perform the updates.

Transaction ID	Transaction Process
M1	Maintains the Message Generation table
M2	Maintains the Message Text table
M3	Maintains the Message Help Text table
M4	Maintains the Message Routing table
M5	Maintains the Screen Help Text table
M6	Maintains the Secondary Transaction Routing table

AUDIT DATA BASE

The Audit Data Base contains the rules that describe the audit/logic processing. It consists of operation descriptors (which control the logic), data descriptors (which contain literals and constants) and optionally table data.

AUDIT DATA BASE (DL/I VERSION)

Audit Data Base Layout (DL/I Version)

Figure 4-19 shows the DL/I structure of this data base. Each audit routine is associated with a field and contains a root segment plus one to three segment paths. These paths coincide with the three audit phases described in the chapter Chapter 5, "Audit Logic Processing" on page 5-1. A fourth path is used for tables of encode/decode values.

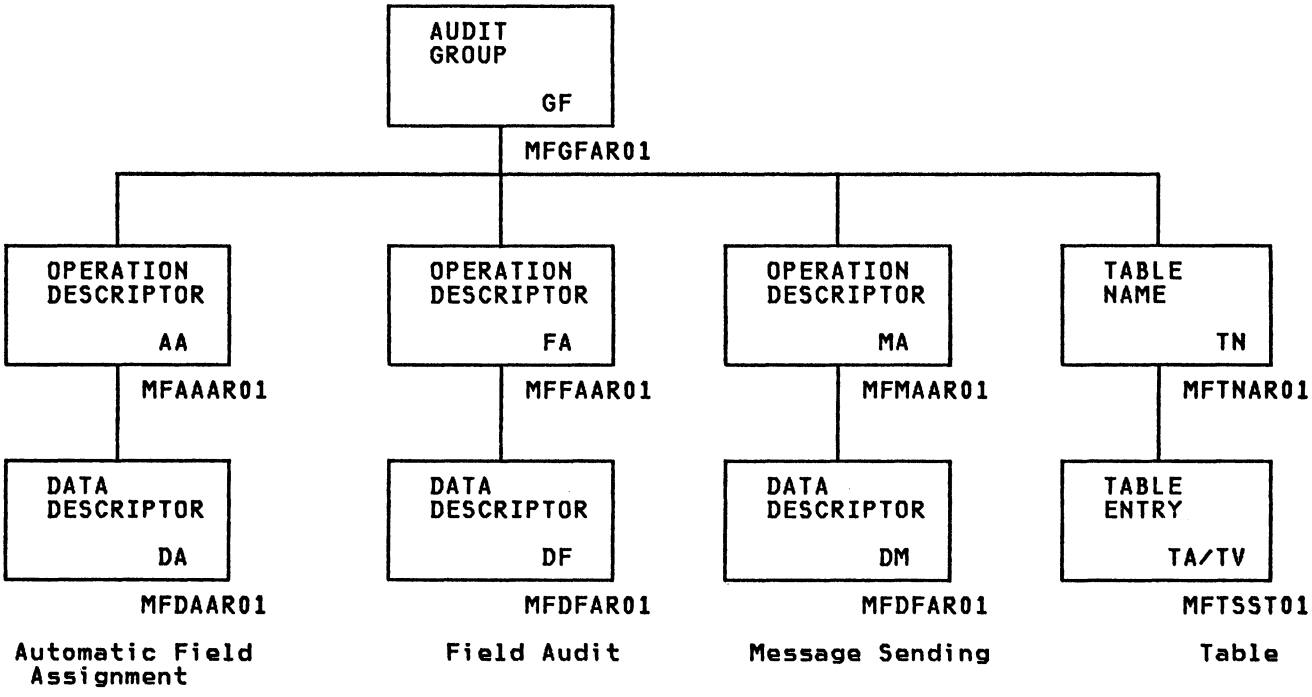


Figure 4-19. Audit Data Base

Each segment layout is described below.

1. Audit Group Segment (GF)

   Segment Length = 16 bytes

Position	Length	Data Type	Data Description
1	16	alphanum	Key of Audit record

   The Audit Group segment (GF) is a 16-byte, key-only segment. The key is constructed in one of five formats.

   Format 1: Used to specify Audit records for Preaudit, Key Audit, Standard Audit and Key Audit (KANAME=ALT) calls and contains an eight-byte Audit group code plus an eight-byte fully qualified field name. The Audit group code is formatted as follows:

   ssssyyyy where

ssss = Application System Identification

yyyy = Audit group - Specified in the Rules Generator operand 'AGROUP'

Since Audit group code is associated with an Input Transaction Rule, a field can have different audit/logic processing dependent upon the Input Transaction Rule currently being processed.

The eight-byte field name consists of:

ssxxffff where

ss = Two high-order bytes of Application System ID

xx = ID of segment that contains the field

ffff = Field ID

This name matches the Segment Layout Rule field name created by Rules Generator input.

Format 2: Used to specify Primary and Secondary key audits and is constructed as follows:

KEYAUDITssxxffff

The first eight bytes are a constant 'KEYAUDIT' followed by an eight-byte field name. Key Audits for a field will reside within a different data base record than Preaudits or Standard Audits for the same field. As discussed earlier, Primary Key Audits reside in the AFA leg and Secondary Key Audits reside in the FAL leg.

Key Audits also can be constructed using format 1. The Auditor will retrieve Key Audit operations using format 1 if KANAME=ALT has been specified on the Input Transaction Rule. If KANAME=ALT is not specified format 2 will be used. Refer to KANAME parameter in Chapter 2, "Rules Generator."

Format 3: Used to specify "common audits". An installation may have fields of the same name used in many segments and applications. An example might be a date field called DATE. The audit/logic processing for this field is constant for all occurrences. The user can designate the field name as a "common field" through Rules Generator input. This is done by specifying CAUDIT=Y each time the field is described to the Rules Generator. This will mark the field entry in the Segment Layout Rule as a Common field. When a field name is specified as common, only one set of audit/logic descriptors need be created and maintained for all of its occurrences. The key for a common field has the following format:

COMMON000000ffff

The first 12 bytes are constant (COMMON000000) and bytes 13-16 contain the Field ID. Upon detection of a common field, the Auditor will construct the root key as shown above.

Format 4: Used to specify audit subroutines. As discussed later, an audit operation is available which allows branching to an audit subroutine from any other audit routine. In this operation, the user specifies the 16-byte root key of the subroutine. Any meaningful 16-byte definition may be used for this key. For example:

SUBROUTINE-#0001
or
0000000000000001

would both represent a valid key. If audit load modules are used, the first eight characters of this format are

restricted to alphabetic or or numeric characters and the first character must be alphabetic.

Format 5: Used to specify an audit table as described under table operations. Any meaningful 16-byte definition may be used for the key. If audit load modules are used, the first eight characters of this format are restricted to alphabetic or numeric characters and the first character must be alphabetic.

2. Operation Descriptor Segments (AA,FA,MA)

Segment Length = 28 bytes

Position	Length	Data Type	Data Description
1	2	alphanum	Segment key
3	2	alphanum	Operation code
5	8	alphanum	Related field name
13	2	alphanum	Next true sequence
15	2	alphanum	Next false sequence
17	4	num	Message or routing number
21	8	alphanum	Reserved

3. Data Descriptor Segments (DA,DF,DM)

Segment Length = 28 bytes

Position	Length	Data Type	Data Description
1	4	num	Segment key
5	24	alphanum	Data values

4. Table Name Segment (TN)

Segment Length = 28 bytes

Position	Length	Data Type	Data Description
1	6	alphanum	Segment key
7	22	alphanum	Table description

5. Table Entry Segment (TA,TV)

Segment Length = 78 bytes

Position	Length	Data Type	Data Description
1	8	alphanum	Table argument
9	70	alphanum	Table function

Updating the Audit Data Base (DL/I Version)

The Audit Data Base can be updated through the Standard Processing and Text Utility Processing function. Audit descriptor segments can be created and maintained through the use of transactions that are supplied with the system. A transaction and its associated rules and screens are supplied for each of the nine segment types in the Audit Data Base.

The transactions (SEG ID) available for selection are as follows:

SEG NAME	SEG ID	TYPE OF SEGMENT	LENGTH	PATH
MFGFAR01	GF	Audit Group/Field	16	Root
MFAAAR01	AA	Operation Descriptor	28	AFA
MFDAAR01	DA	Data Descriptor	28	AFA
MFFAAR01	FA	Operation Descriptor	28	Field Audit
MFDFAR01	DF	Data Descriptor	28	Field Audit
MFMAAR01	MA	Operation Descriptor	28	Msg. Send
MFDMAR01	DM	Data Descriptor	28	Msg. Send
MFTNAR01	TN	Table Name	28	Table
MFTAAR01	TA	Table Entry	78	Table
	TV	Re-defines the Table Entry segment with the value field as the key		

These segment names and IDs are associated with the Audit Data Base shown in Figure 4-19.

To access the Audit Data Base with Standard Processing, the user enters option 'D' (Transaction Selection), the desired transaction mode (1-6), and a segment ID (GF, AA, DA, FA, DF, MA, DM, TN, TA, or TV) to the Primary Option Menu screen. Key selection can help the terminal user locate the desired segment.

The following is an example of how to add segments to the Audit Data Base. The example explains the steps and screen interactions required to add the three major segment types of this data base:

Audit Group/Field Name (GF)

Field Audit Operation Descriptor (FA)

Field Audit Data Descriptor (DF)

The example sets up descriptor segments to audit a field named MFCDCLOR for a value between 500 and 600. If the field does not fall within this range, error message number 9100 will be displayed.

The first segment added is the root segment for the Audit Group and field to be audited. On the Primary Option Menu, the user selects:

- Option 'D'
- Transaction mode '2' (INITIATE)
- Segment ID 'GF'

The screen that will be displayed as a result of this entry is shown in Figure 4-20. On this screen the user must enter the primary key for the segment.

```
                    A U D I T   D A T A   B A S E
          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
ADD                        TRANSACTION: AUDIT GROUP/FIELD
OPTION:     TRX: 4GF  KEY:
                  ** ENTER THE FOLLOWING KEY INFORMATION **
        SYSTEM ID/AUDIT GROUP-
        FIELD NAME (SSXXFFFF)-
```

Figure 4-20. Primary Key Selection Screen (display)

To this screen the user enters the Audit Group/Field, which is a 16-character field. For this example, the Audit Group is 'MFC1MAUD' and the field name is 'MFCDCLOR'. Figure 4-21 shows the field entered to the screen.

```
                    A U D I T   D A T A   B A S E
          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
ADD                        TRANSACTION: AUDIT GROUP/FIELD
OPTION:     TRX: 4GF  KEY:
                  ** ENTER THE FOLLOWING KEY INFORMATION **
        SYSTEM ID/AUDIT GROUP- MFC1MAUD
        FIELD NAME (SSXXFFFF)- MFCDCLOR
```

Figure 4-21. Primary Key Selection Screen (as entered)

When this screen is entered, the system redisplays the screen shown in Figure 4-22. This screen is redisplayed to allow data other than the primary key to be entered.

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

ADD                          TRANSACTION: AUDIT GROUP/FIELD
OPTION:     TRX: 4GF  KEY:   MFC1MAUDMFCDCLOR
   *** ENTER DATA FOR ADD ***
        SYSTEM ID/AUDIT GROUP- MFC1MAUD
        FIELD NAME (SSXXFFFF)- MFCDCLOR
```

Figure 4-22. Screen to Add Other Than Primary Key Data

Since the root segment contains no fields other than the primary key, the reentry of the screen will cause the segment to be added to the data base.

Once a root segment is available, field audit Operation Descriptor segments can be created. The user enters the following to the Primary Option Menu:

- Option 'D'
- Transaction Mode 4 (ADD)
- Segment ID 'FA'

The screen shown in Figure 4-23 is displayed for entry of the primary key information for an Operation Descriptor segment.

```
                    A U D I T   D A T A   B A S E
          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
ADD                       TRANSACTION: FIELD AUDIT OPERATION DESC
OPTION:     TRX: 4FA  KEY:
                 ** ENTER THE FOLLOWING KEY INFORMATION **
        SYSTEM ID/AUDIT GROUP-
        FIELD NAME (SSXXFFFF)-
        SEGMENT SEQ-----------
```

Figure 4-23. Primary Key Selection Screen for Operation Descriptor Segment

The entries to this screen consist of the Audit Group/Field of the root segment and the sequence number for this Operation Descriptor. In this example, this is the first Operation Descriptor for the field. Figure 4-24 illustrates the screen with the appropriate entries.

```
                    A U D I T   D A T A   B A S E
          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
ADD                       TRANSACTION: FIELD AUDIT OPERATION DESC
OPTION:     TRX: 4FA  KEY:
                 ** ENTER THE FOLLOWING KEY INFORMATION **
        SYSTEM ID/AUDIT GROUP- MFC1MAUD
        FIELD NAME (SSXXFFFF)- MFCDCLOR
        SEGMENT SEQ----------- 01
```

Figure 4-24. Operation Descriptor Information Entered

When this screen is entered to the system, a new screen is displayed to allow descriptor fields to be added. Figure 4-25 shows this screen.

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

ADD                          TRANSACTION: FIELD AUDIT OPERATION DESC
OPTION:    TRX: 4FA  KEY:   MFC1MAUDMFCDCLOR01
   *** ENTER DATA FOR ADD ***
        SYSTEM ID/AUDIT GROUP- MFC1MAUD
        FIELD NAME (SSXXFFFF)- MFCDCLOR
        SEGMENT SEQ----------- 01
        DESCRIPTOR CODE------- __
        RELATED FIELD--------- ________
        NEXT TRUE SEQ NO------ __
        NEXT FALSE SEQ NO----- __
        MESSAGE #------------- 0000
```

Figure 4-25. Screen for Adding Descriptor Fields

For the present example, the data entered will be descriptor code 21, a next true sequence number of 00, and message number 9100 as illustrated in Figure 4-26. Since the Next False Sequence Number field is blank, a false condition will cause message 9100 to be displayed during auditing. Entry of this screen will add the segment to the Audit 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:   MFC1MAUDMFCDCLOR01
   *** ENTER DATA FOR ADD ***
        SYSTEM ID/AUDIT GROUP- MFC1MAUD
        FIELD NAME (SSXXFFFF)- MFCDCLOR
        SEGMENT SEQ----------- 01
        DESCRIPTOR CODE------- 21
        RELATED FIELD--------- ________
        NEXT TRUE SEQ NO------ 00
        NEXT FALSE SEQ NO----- __
        MESSAGE #------------- 9100
```

Figure 4-26. Descriptor Fields as Entered

Descriptor code 21 specifies that values in a Data Descriptor segment are needed. Therefore, a Data Descriptor segment or segments must be added to the data base. To accomplish this, the user enters the following to the Primary Option Menu:

- Option 'D'
- Transaction Mode 4 (ADD)
- Segment ID 'DF'

The screen shown in Figure 4-27 is displayed. On this screen, the user enters primary key information for the Data Descriptor segment.

```
                    A U D I T   D A T A   B A S E
          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
ADD                      TRANSACTION: FIELD AUDIT DATA DESCRIPTOR
OPTION:     TRX: 4DF  KEY:                      00
                 ** ENTER THE FOLLOWING KEY INFORMATION **
        SYSTEM ID/AUDIT GROUP-
        FIELD NAME (SSXXFFFF)-
        SEGMENT SEQ-----------
        DATA SEQ-------------- 00
```

Figure 4-27. Data Descriptor Entry Screen

The fields are entered as shown in Figure 4-28. Since this will be the first Data Descriptor segment under Operation Descriptor 01, the key field should be 0001.

```
                    A U D I T   D A T A   B A S E
          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
ADD                       TRANSACTION: FIELD AUDIT DATA DESCRIPTOR
OPTION:    TRX: 4DF  KEY:                        00
                 ** ENTER THE FOLLOWING KEY INFORMATION **
        SYSTEM ID/AUDIT GROUP- MFC1MAUD
        FIELD NAME (SSXXFFFF)- MFCDCLOR
        SEGMENT SEQ----------- 01
        DATA SEQ-------------- 0001
```

Figure 4-28. Data Descriptors as Entered

Entry of this screen causes the screen shown in Figure 4-29 to be displayed. This screen allows the user to enter data values into the segment.

```
                    A U D I T   D A T A   B A S E
          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
ADD                       TRANSACTION: FIELD AUDIT DATA DESCRIPTOR
OPTION:    TRX: 4DF  KEY:   MFC1MAUDMFCDCLOR010001
   *** ENTER DATA FOR ADD ***
        SYSTEM ID/AUDIT GROUP- MFC1MAUD
        FIELD NAME (SSXXFFFF)- MFCDCLOR
        SEGMENT SEQ----------- 01
        DATA SEQ-------------- 0001
        DATA------------------ ______________________
```

Figure 4-29. Screen for Adding Data Values to Segment

In this example, the value range was between 500 and 600. These values are entered in the data field as shown in Figure 4-30.

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

ADD                          TRANSACTION: FIELD AUDIT DATA DESCRIPTOR
OPTION:     TRX: 4DF  KEY:   MFC1MAUDMFCDCLOR010001
   *** ENTER DATA FOR ADD ***
        SYSTEM ID/AUDIT GROUP- MFC1MAUD
        FIELD NAME (SSXXFFFF)- MFCDCLOR
        SEGMENT SEQ----------- 01
        DATA SEQ-------------- 0001
        DATA------------------ (500,600)
```

Figure 4-30. Data Values as Entered

Entry of this screen to the system causes the Data Descriptor segment to be added to the data base.

The method for adding segments to the Automatic Field Assignment, Message Sending, and Table legs of the Audit Data Base is essentially the same as that shown in the example above.

Adding Table leg entries can be accomplished as follows:

- Add a Table Name segment (TN) using Standard Processing
- Table Entry segments (TA) may be added through Standard processing (1 segment at a time), or Text Utility processing, (up to 15 segments on one screen). In either case, the rules and screens are provided and accessed through Primary and Secondary Options Menu selection. Refer to the IMS Application Development Facility II Version 2 Release 2 User Reference manual for more information on text utility processing.

Audit Data Base Layout (DB2 Version)

In the DB2 environment, the Audit Data Base consists of three DB2 tables. Each table is formatted as follows:

Table ID	Table content
A1	Operation descriptors
A2	Data descriptors
A3	Encode/decode table data

1. Operation Descriptor table (A1)

Position	Length	Data Type	Data Description
1	8	alphanum	Audit Group (key)
9	8	alphanum	Field name (key)
17	2	alphanum	Phase sequence (key)
19	2	alphanum	Sequence (key)
21	2	alphanum	Operation code
23	8	alphanum	Related field name
31	2	alphanum	Next true sequence
33	2	alphanum	Next false sequence
35	4	num	Message or routing number
39	8	alphanum	Reserved

2. Data Descriptor table (A2)

Position	Length	Data Type	Data Description
1	8	alphanum	Audit Group (key)
9	8	alphanum	Field name (key)
17	2	alphanum	Phase sequence (key)
19	2	alphanum	Sequence (key)
21	4	num	Sequence (key)
25	24	alphanum	Data values

3. Encode / Decode table (A3)

Position	Length	Data Type	Data Description
1	8	alphanum	Audit Group (key)
9	8	alphanum	Field name (key)
17	6	alphanum	Table identifier (key)
23	8	alphanum	Table argument
31	70	alphanum	Table function

Updating the Audit Data Base (DB2 Version)

The three Audit Data Base tables can be updated through Standard Processing. Audit descriptors and encode/decode table entries can be maintained through transactions that are supplied with the system. A transaction and its associated rules and screens are supplied for each of the three tables that make up the Audit Data Base.

An authorized user signs on to IMSADF II and selects Option D, Standard Segment processing. The flow of screens will look like those shown in the section titled "Updating the Audit Data Base (DL/I Version)". Some screens will be slightly different. The following transaction IDs should be selected to perform the updates.

Transaction ID	Transaction Process
A1	Maintains the Operation Descriptor table
A2	Maintains the Data Descriptor table
A3	Maintains the Encode/Decode table entries

UPDATING (DL/I) DYNAMIC RULES DATA BASES IN BATCH MODE

Data bases can be updated in the batch or BMP environment using the Batch Driver. To execute the Batch Driver with the Data Base Update Special Processing Routine, the procedure MFCB (member in IMSADF.JCLLIB.JCL) can be used. The PSB for updating the data bases is MFC1BCTP and is provided in the PSB source and load libraries.

Input to the Batch Driver is an OS/VS sequential file with a DDNAME of TRANSIN. Each record consists of a transaction name and the data for processing the transaction. If the transaction encompasses more than one 80-byte card image, the continuation records do not contain the transaction name. The format of the transaction name is

```
????Bmxx
```

where m = the mode, 1 to 6

xx = the target segment identification

The transaction name must appear in columns 1 through 8. Different transactions can be included in a single batch run. The following layout of card images define the transaction data for updating IMSADF II data bases.

Sign-on Profile Data Base

Card	Column	Length	PG - Project/Group Segment
1	9	2	Project/Group identification
	11	26	Description of PG function
	37	4	Application System Identification

EX. MFC1B2PGQQSAMPLE CHECKOUT (11 blanks) SAMP

			SR - Employee Userid Segment
1	9	2	Key of Project/Group Segment
	11	6	Employee Userid
	17	11	Employee name
	28	2	Profile ID
	30	8	INFO (optional lockword)

EX. MFC1B4SRQQ999999J.SMITH AB

			PR - Profile Authority Segment
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 through 60) on card number 2, 3, 4, 5, and 6.)

	1	2	Transaction ID
	3	1	Level of authority (1 to 6)
	4	1	Reserved
2	1	60	Transaction IDs and authority
3	1	60	level as indicated above
4	1	60	
5	1	60	
6	1	60	

A $$ is needed to indicate end of data if less than 21 cards are specified.

```
EX.  MFC1B4PRQQAB005
     HD10SY30PG10SR30PR30 $$
```

Audit Data Base

Card	Column	Length	GF - Audit Group Segment
1	9	16	Audit Group Key (Audit Group name & fieldname)

EX. MFC1B2GFSAMPYYYYSAPDINVC

AA - Automatic Field Assignment Operation Descriptor

1	9	16	Key of GF Segment
	25	2	Sequence number of AA Segment
	27	2	Operation Descriptor
	29	8	Related field
	37	2	Next true AA Segment
	39	2	Next false AA Segment
	41	4	Message number

EX. MFC1B4AASAMPYYYYSAPDINVC0112 00 9998

DA - Automatic Field Assignment Data Descriptor

1	9	16	Key of GF segment
	25	2	Key of AA segment
	27	4	Sequence number of DA segment
	31	24	Data values

EX. MFC1B4DASAMPYYYYSAPDINVC010001 (H)

FA - Field Audit Operation Descriptor

1	9	16	Key of GF Segment
	25	2	Sequence number of FA Segment
	27	2	Operation Descriptor
	29	8	Related Field
	37	2	Next true FA Segment
	39	2	Next false FA Segment
	41	4	Message number

EX. MFC1B4FASAMPYYYYSACDDIPU0132 00 9999

DF - Field Audit Data Descriptor

1	9	16	Key of GF segment
	25	2	Key of FA segment
	27	4	Sequence number of DF segment
	31	24	Data values

EX. MFC1B4DFSAMPYYYYSACDDIPU010001(P,U)

MA - Message Sending Operation Descriptor

1	9	16	Key of GF segment
	25	2	Sequence number of MA segment
	27	2	Operation Descriptor
	29	8	Related field
	37	2	Next true MA segment
	39	2	Next false MA segment
	41	4	Message group code

EX. MFC1B4MASAMPYYYYSAIVSTCK0105 00 0080

DM - Message Sending Data Descriptor

1	9	16	Key of GF segment
	25	2	Key of MA segment
	27	4	Sequence number of DM segment
	31	24	Data values

EX. MFC1B4DMSAMPYYYYSAIVSTCK010001(100)

TN - Table Name

1	9	16	Key of GF segment
	25	6	Key of TN segment
	31	22	Table description

EX. MFC1B4TNSAMPYYYYSACDTABLTABLE1THIS IS TABLE#1

TA - Table Entry

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

EX. MFC1B4TASAMPYYYYSACDTABLTABLE11234
This is the value for argument 1234

Message Data Base

Card	Column	Length	AH - Auto Message Sending Header
1	9	8	Conditions for auto message sending

EX. MFC1B2AHM004IY05

AR - Auto Message Routing

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	4	User to receive message

EX. MFC1B4ARM004IY0500000001MX0901 (16blanks) 999999

HD - Message Generation Header

Card 1 has the HD transaction name MFC1B2HD.

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	.

EX. MFC1B2HD
SAMP99990070SACDDIPU030

SY - Message text

1	9	8	Key of HD segment
2	1	8	Sequence number of SY segment
	9	70	Message text

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

SD - Secondary Transaction Destination

1	9	8	Output Format Rule name (Key)
	17	8	Default destination
	25	62	Comments for user information (continued on 2nd card). $$ is needed on first card

if the second card is not used.

EX. MFC1B23DMFORPD01IOPCB ERRORS SECONDARY XACT $$
LABOR

LT - Logical Terminal segment

1	9	8	Key of SD segment
	17	8	Entering terminal (Key)
	25	8	Receiving terminal #1
	33	8	Receiving terminal #2
	41	8	Receiving terminal #3
	49	8	Receiving terminal #4
	57	8	Receiving terminal #5
	65	8	Receiving terminal #6
	73	8	Receiving terminal #7
2	1	8	Receiving terminal #8

A $$ is needed on the first card if the second card is not used.

EX. MFC1B4LTMFORPD01L3277099L3286001IOPCB $$

UH - User Header segment

1	9	8	Userid (Key)
	17	22	User's name

EX. MFC1B2UH999999 BJ.SMITH

Card	Column	Length	HE - Help Text Header
1	9	8	Key of HE segment

EX. MFC1B2HESAMPES0@

Card	Column	Length	HT - Screen Help Text Header
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

EX. MFC1B4HTSAMPES0@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$$$$.

HT segments will be displayed exactly as the 79 byte lines of help text are built.

The end of data characters (installation defined) should be specified twice if cards 4 through 22 are not needed. At execution time, this indicates to the conversational driver only the amount of valid data to pass to the HELP screen.

Card	Column	Length	MH - Message Help Text Header
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

EX. MFC1B4MHSAMP999900000010001

THIS IS LINE 1 OF THE ERROR HELP TEXT.

THIS IS THE LAST LINE OF THE ERROR HELP TEXT$$$$.

The end of data characters (installation defined) should be specified twice if cards 3 or 4 are not needed.

UPDATING (DB2) DYNAMIC RULES DATA BASES IN BATCH MODE

DB2 tables can be updated in the BMP environment using the Batch Driver. To execute the Batch Driver with the Data Base Update Special Processing Routine, the procedure MFCB (member in IMSADF.JCLLIB.JCL) can be used. The PSB for updating the data bases is MFC1BCTP and is provided in the PSB source and load libraries.

Input to the Batch Driver is an OS/VS sequential file with a DDNAME of TRANSIN. Each record consists of a transaction name and the data for processing the transaction. If the transaction encompasses more than one 80-byte card image, the continuation records do not contain the transaction name. The format of the transaction name is

????Bmxx

where m = the mode, 1 to 6

xx = the target segment identification

The transaction name must appear in columns 1 through 8. Different transactions can be included in a single batch run. The following layout of card images define the transaction data for updating the DB2 tables.

Sign-on Data Base (three tables)

Card	Column	Length	S1 - Project/Group Table
1	9	2	Project/Group identification
	11	26	Description of PG function
	37	4	Application System Identification

EX. MFC1B2S1QQSAMPLE CHECKOUT (11 blanks) SAMP

Card	Column	Length	S2 - Employee Userid Table
1	9	2	Project/Group
	11	6	Employee Userid
	17	11	Employee name
	28	2	Profile ID
	30	8	INFO (optional lockword)

EX. MFC1B4S2QQ999999J.SMITH AB

Card	Column	Length	S3 - Profile Authority Table
1	9	2	Project/Group
	11	2	Profile ID
	13	3	Number of Transaction IDs

(The following four positions are repeated 15 times per card (columns 1 through 60) on card numbers 2 through 21.)

	1	2	Transaction ID
	3	1	Level of authority (1 to 6)
	4	1	Reserved
2	1	60	Transaction IDs and authority
3	1	60	level as indicated above
4	1	60	
5	1	60	
6	1	60	
7	1	60	
8	1	60	
9	1	60	
10	1	60	
11	1	60	
12	1	60	
13	1	60	
14	1	60	
15	1	60	
16	1	60	
17	1	60	
18	1	60	
19	1	60	
20	1	60	
21	1	60	

A $$ is needed to indicate end of data if less than 21 cards are specified.

```
EX.  MFC1B4S#QQAB005
     HD10SY3OPG10SR3OPR30 $$
```

Audit Data Base (three tables)

Card	Column	Length	A1 - Operation descriptor table
1	9	16	Audit Group Key (Audit Group name & fieldname)
	25	2	phase (AA,FA,MA)
	27	2	Sequence number of descriptor
	29	2	Operation Descriptor
	31	8	Related field
	39	2	Next true AA Segment
	41	2	Next false AA Segment
	43	4	Message number

```
EX.  MFC1B4A1SAMPYYYYSAPDINVCAA0112        00  9998
```

Card	Column	Length	A2 - Data descriptor table
1	9	16	Audit Group Key (Audit Group name & fieldname)
	25	2	phase (AA,FA,MA)
	27	4	Sequence number of descriptor
	31	24	Data values

```
EX.  MFC1B4A2SAMPYYYYSAPDINVCAA010001 (H)
```

Card	Column	Length	A3 - Encode/decode table
1	9	16	Audit Group Key (Audit Group name & fieldname)
	25	6	Key of TN segment
	31	8	Key of TA segment (argument)
2	1	70	Table value

```
EX.  MFC1B4A3SAMPYYYYSACDTABLTABLE11234
     This is the value for argument 1234
```

Message Data Base (seven tables)

Card	Column	Length	M1 - Message Generation

Card 1 has the A1 transaction name MFC1B2A1.

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	.

EX. MFC1B2HD
SAMP99990070SACDDIPU030

Card	Column	Length	M2 - Message Text
1	9	8	message number (key)
2	1	8	sequence number (00000001,00000002,,)
	9	70	Message text

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

Card	Column	Length	M3 - Message Help Text
1	9	8	Key of message
	17	4	sequence key (0001,0002,,,)
2	1	79	Text for Message HELP
3	1	79	Text for Message HELP
4	1	79	Text for Message HELP

EX. MFC1B4MHSAMP9999000000010001

THIS IS LINE 1 OF THE ERROR HELP TEXT.

THIS IS THE LAST LINE OF THE ERROR HELP TEXT$$$$.

The end of data characters (installation defined) should be specified twice if cards 3 or 4 are not needed.

Card	Column	Length	M4 - Automatic message sending
1	9	8	message sending key
	17	8	sequence key (00000001,00000002,,,)
	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	4	User to receive message

EX. MFC1B4M4M004IY0500000001MX0901 (16 blanks) 999999

Card	Column	Length	M5 - Screen Help Text
1	9	8	Key of screen
	17	4	sequence key (0001,0002,,,)
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

EX. MFC1B4M5SAMPES0@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$$$$.

The end of data characters (installation defined) should be specified twice if cards 4 through 22 are not needed. At execution time, this indicates to the conversational driver only the amount of valid data to pass to the HELP screen.

Card	Column	Length	M6 - Secondary Transactions
1	9	8	Output Format Rule name (Key)
	17	8	Default destination
	25	62	Comments for user information (continued on 2nd card). $$ is needed on the first card if the second card is not used.
2	7	8	Entering terminal (Key)
	15	8	Receiving terminal #1
	23	8	Receiving terminal #2
	31	8	Receiving terminal #3
	39	8	Receiving terminal #4
	47	8	Receiving terminal #5
	55	8	Receiving terminal #6
	63	8	Receiving terminal #7
	71	8	Receiving terminal #8

A $$ is needed on the first card if the second card is not used.

EX. MFC1B2M6MFORPD01IOPCB ERRORS ON SECONDARY XACT $$

or

EX. MFC1B2M6MFORPD01IOPCB ERRORS ON SECONDARY XACT
L3277099L3286001IOPCB $$

Card	Column	Length	M7 - User Mailbox
1	9	8	Userid (Key)
	17	22	User's name

EX. MFC1B2UH999999 BJ.SMITH

There are special transaction ids set up to allow the deletion of rows within a table. For example; to delete a Project/Group and its associated userids and profiles under DL/I you delete the PG root segment. Under DB2 deletes must be made against three tables. The following transactions are available for delete only.

Transaction ID	Transaction Process
S6	delete all userids under a Project /Group
S7	delete all profiles under a Project /Group
S8	delete all messages under a Project /Group
A5	delete all Operation descriptors under an audit
A6	delete all Data descriptors under an audit
A7	delete all Encode/decode entries under an audit
A8	delete all Data descriptors under an Operation descriptor
A9	delete all Encode/decode entries under a table
M0	delete all help text for a screen
MC	delete all message text for a message
MD	delete all help text for a message
ME	delete all message routing entries for a key

Following is an example of each delete transaction.

Transaction ID	Example	Key
S6	MFC1B3S6QQ	project group
S7	MFC1B3S7QQ	project group
S8	MFC1B3S8QQ	project group
A5	MFC1B3A5SAMPSACDCLOR	audited field
A6	MFC1B3A6SAMPSACDCLOR	audited field
A7	MFC1B3A7SAMPSACDCLOR	audited field
A8	MFC1B3A8SAMPSACDCLORAA01	field/Op descriptor
A9	MFC1B3A7SAMPSACDCLORTABLE1	field/table name
M0	MFC1B3M0SAMPESM@	screen key
MC	MFC1B3MCSAMP2001	message number
MD	MFC1B3MDSAMP2001	message number
ME	MFC1B3MESQQ5CD01	routing key

CHAPTER 5. AUDIT LOGIC PROCESSING

IMSADF II uses the term 'auditing' to describe the execution and control of procedural logic operations. This auditing function is managed by a combination of a precoded Auditor module and generated audit operations. The developer defines and generates these operations through the Audit Language, which is described in Chapter 3, "Audit Language."

This chapter describes how the Auditor executes the operations generated from the Audit Language. Chapter 3, "Audit Language" references sections within this chapter, when a more detailed description of an operation is required.

Functions that are available through audit/logic operations are:

- Full comparison capability
  - Arithmetic
  - Logical
  - Equal, greater than, less than
  - Range checking
- Arithmetic capability
  - Add, subtract, multiply, divide
- Field Assignment
  - One field to another
  - Constant to a field
- Screen Attribute Modification
- Automatic Message Sending
  - Error messages (field does not pass validation)
  - Informative messages (notify a specific user or project group of transaction results.)
  - Warning messages
  - Secondary Transactions
- Encode/Decode
  - Related field to audited field
  - Audited field to related field
- DL/I Calls
- DB2 Calls

The Facility provides many pre-written operation routines that may be combined to form a complex audit/logic routine. In addition, user-written audit/logic routines may be added to the system if a unique requirement exists.

THE AUDITOR

The Auditor module controls all audit/logic operations within the system. This module is called by either a Transaction Driver (Conversational, Nonconversational, or Batch) or a Special Processing Routine.

The Auditor can be invoked at four points within the transaction. They are:

1. Primary Key Selection - immediately after the key information is entered. This point is used to process keys prior to segment retrieval.

2. Secondary Key Selection - immediately after accessing a segment to be displayed on the Secondary Key Selection screen (Conversational only). This point can be used to prohibit the display of specific segments on the Secondary Key Selection screen or to eliminate access of additional segments.

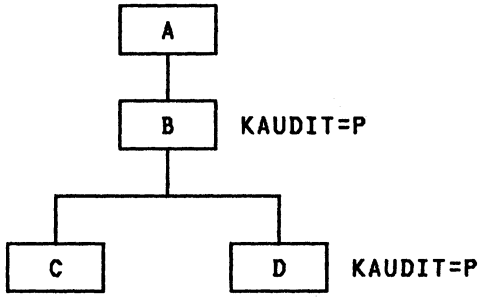
3. Preaudit - immediately after the DBPATH segment(s) have been loaded in the SPA area. This point is used to validate and process data prior to screen display or batch update.

4. Standard Audit - immediately after the segment data has been modified from the screen or batch input. This point is used to validate and process data prior to applying the updates to the physical record in the data base. If the transaction is Special Processing the Standard Audit call must be made from the Special Processing routine as described in Chapter 6, "Special Processing and User Exits."

Once the Auditor is invoked, it will determine (through internal controls) the transaction point at which it was called. The Segment Layout Rule(s) and/or Input Transaction Rule are then used to determine the Audit routines to invoke. These routines, which are described later, reside in the Audit Data Base (Figure 4-19) or, optionally, as static audit load modules.

PRIMARY KEY SELECTION CALL

The Auditor is called during Primary Key Selection if a loadable segment in the DBPATH(s) has a key field specified with KAUDIT = P or Y. After key information is entered (i.e., through the Primary Key Selection, Primary Option, Secondary Option, Segment Display screens or Batch input), a segment retrieval occurs to the segment with a key field specified as KAUDIT. The Auditor is then called and executes the audit operations defined for key fields within that segment, which are specified KAUDIT = P or Y. If additional segments have the KAUDIT parameter, the process is repeated. For example:



DBPATH=(C,D)

1. The key data entered for the two paths is read and mapped into the concatenated key area for each segment. The key data for segments B and D is also mapped into the segment I/O areas for the two segments.

2. Segment A is retrieved by a Get Unique (GU) or through Secondary Key Selection.

3. The Auditor is called and all key fields within segment B with KAUDIT=P or Y are audited.

4. Upon return from the Auditor, the (possibly updated) key fields in segment B are used to retrieve the segment by a 'GU' or through Secondary Key Selection.

5. Segment C is retrieved as described under Steps 1 and 2.

6. Segment D is retrieved as described under Steps 3 and 4. The rules for acd mode and insert eligibility apply to segments processed through Key Audit in the same manner as segments which are not audited. If a key field is audited at Primary Key Selection and subsequently Secondary Key Selection is used to select the segment, the Auditor is recalled after the segment is selected. All Primary Key Audits will be reprocessed against the selected segment's key data. This means if segment B in the above example is retrieved in step 4 through Secondary Key Selection, step 3 will be reprocessed.

Note: If any DBPATH segments are retrieved through an audit operation, their segment retrieved flags should be turned off before returning. The key Selection module will attempt to retrieve all DBPATH segments and does not expect them to be previously retrieved.

For keyaudit processing, the Auditor verifies that:

- A field designated as ALPHA contains only alphabetic characters or blanks.
- A field designated as NUMERIC contains only numeric characters.
- A field designated as DATE contains a valid month (1-12) and a valid day for the month (e.g. JUN 1-30, Dec. 1-31).

If the Auditor encounters an error during Primary Key Audit, the Primary Key Selection screen is displayed with either the error message or a message specifying that an error has occurred. An 'E' entered in the OPTION field will display the Error screen. Upon reentry, control is returned to the Primary Key Selection screen. The Primary Key Selection Auditor call is normally used to build or verify keys by use of constant data, encode/decode operations, or complement/uncomplement processing. Pseudo segment field(s) can be linked with the key field(s) to hold the form entered and displayed to the terminal operator, if it is different from the data base form (see the COFIELD operand in Chapter 2, "Rules Generator" for additional information). This, for example, allows the operator to enter and display a date in normal format while the actual data base key is in a complemented format. Data entered into a key field linked to a COFIELD will cause the data to be mapped into the specified COFIELD instead of the segment I/O area. The Auditor call can be used to process the COFIELD data and create the data base form in the segment I/O area. Whether or not a COFIELD is used, the Primary Key Audit is triggered from a key field defined as KAUDIT = P or Y. The Primary Key Selection call is valid in Conversational, Nonconversational, and Batch processing.

SECONDARY KEY SELECTION CALL

During Secondary Key Selection, the Auditor is called if the segment contains a key or related field specified as KAUDIT = S or Y. The Auditor is called as each segment occurrence is loaded and will execute defined audit operations on all key or related fields within the segment which specify KAUDIT = S or Y. If an error is detected, the Primary Key Selection screen is displayed with a message specifying that an error has occurred. An 'E' entered in the OPTION field will display the Error screen. Upon reentry, control is returned to the Primary Key Selection screen.

For keyaudit processing, the Auditor verifies that:

- A field designated as ALPHA contains only alphabetic characters or blanks.
- A field designated as NUMERIC contains only numeric characters.
- A field designated as DATE contains a valid month (1-12) and a valid day for the month (e.g. JUN 1-30, Dec. 1-31).

The Secondary Key Selection call can be used to prohibit display of specific segment occurrences on the Secondary Key Selection screen. An audit operation can be specified which causes the segment being audited to not be displayed. Secondary Key Selection can also be terminated through another operation. This causes the retrieval of segments to

stop and the Secondary Key Selection screen to be displayed. The segment causing the 'stop' operation will not be displayed on the screen.

The Secondary Key Selection call is also used to format the key and related field data. This is accomplished through a COFIELD link as explained above. Since the COFIELD contains the displayable form of the key or related field, the Secondary Key Audits process the data base form of the data and create the COFIELD. This is the reverse of the type of processing explained under Primary Key Audits. Since COFIELD represents both the input form (Primary Key Selection) and output form (Secondary Key Selection) of a key field, it must be used only when the field is flagged for both audit calls (KAUDIT=Y). If a COFIELD is linked with the key field of a segment, selected from the Secondary Key Selection screen, the Auditor is called a final time. This is done to allow conversion of the selected segment's key to its COFIELD format. A Secondary Key Selection related field (RELATED=YES) can also be linked to a COFIELD and can be used to control the form of the related data displayed on the Secondary Key Selection screen. COFIELD's can also be displayed and modified on the Segment Display Screen. This allows the displayable form of the key to be carried forward for display and/or modification. When a COFIELD which is linked to a key field is modified on a Segment Display screen, it is processed as if the key field had been modified.

The Secondary Key Selection call is valid for Conversational processing only. Both the Primary and Secondary Key Selection calls are controlled by the KAUDIT operand in the Input Transaction Rule. Therefore, the Audit calls are made only for transactions that require it and not necessarily each time the segment is used. Refer to 'Audit/Logic Rules" in this chapter for specific information on creation of the Audit routines.

PREAUDIT CALL

After DBPATH segments have been loaded and prior to Segment screen display (conversational processing), the Auditor may be called for Preaudit. In Batch and Nonconversational processing this call occurs after DBPATH segments have been loaded and prior to updating the fields with entered data. The call is made if any field in the transaction is specified with PAUDIT = Y. The Auditor will check the Input Transaction Rule for all fields flagged PAUDIT = Y. The P1 phase is used to control the standard preaudit logic. This includes editing and error processing. If mailbox messages (to a project group or userid) are sent, the P2 phase is also used. The specification of MSG = Y on the field causes the P2 phase to be processed.

If the Auditor encounters any errors or warning messages, the ERROR screen is displayed. Upon reentry, control is returned to the Primary Key Selection screen. The Preaudit call may be used to perform DL/I calls on non-DBPATH segments (TSEGS), and computational or edit logic prior to screen display.

STANDARD AUDIT

After fields have been updated from the screen or batch input, the Standard Audit call is made. If the transaction is Standard Processing the call is made; if Special Processing, the call is bypassed and the special processing routine is invoked. In the latter case, the Special Processing routine must call the Auditor if required.

The Auditor module processes each field through <u>three major phases</u>. These are known as:

- Automatic Field Assignment (P0)
- Field Audit/Logic (P1)
- Message Sending Logic (P2)

Each phase contains the audit/logic operations generated by the Audit Language under a phase identifier (P0, P1, P2). Access to this audit/logic is controlled by parameters in the Input Transaction Rule (ITR) and the Segment Layout Rule(s) (SLR). The following parameters

are checked as the Auditor processes through the three phases. Remember, a Segment Layout Rule audit parameter is present whenever a field is used in any transaction while an Input Transaction Rule parameter pertains only to that transaction.

Automatic Field Assignment Phase

The first phase to be processed is Automatic Field Assignment. It is specified as P0 in the Audit Language coding. This routine controls audit/logic requirements for a specific transaction. It is recommended that all transaction unique logic be processed under this phase. Since each transaction is described and controlled by an Input Transaction Rule, this rule is used to determine if any Audit/Logic processing is required. The following specifications are checked for each field defined in the Auditor section of the Input Transaction Rule, and processed as shown. All Input Transaction Rule specifications are created through Rules Generator input.

SPECIFICATION: Non-initialized field value required by this transaction

RULES GENERATOR OPERAND: REQUIRED=Y

ACTION: The field is checked to determine if it contains an initialized value. If it does, it is marked in error and assigned an error message. This parameter is ignored on a Preaudit call. Initialized values are determined as follows:

Field Type	Initialized Value
ALPHA	Blanks
ALPHA-NUMERIC	Blanks
DECIMAL	Zeros
PACKED DECIMAL	Zeros
BINARY	Zeros
DATE	Blanks or Zeros
HEXADECIMAL	Zeros
BIT	Zero

SPECIFICATION: Audit should be forced on this transaction

RULES GENERATOR OPERAND: FAUDIT=Y

ACTION: The field will be marked as changed in the SLR. Further processing will occur when the field is checked in the Field Audit/Logic phase. AUDIT=Y must also be specified if auditing is to occur in the Field Audit/Logic routine. If phase 0 processing is used for all transaction unique logic (as recommended), this parameter need not be used. This parameter is ignored on a Preaudit call.

In transaction mode 5, process audits will only be invoked if the user has entered amendments on the data display screen. The process phase 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.

SPECIFICATION: Automatic Field Assignment processing required

RULES GENERATOR OPERAND: AFA=Y

ACTION: The phase P0 operations for this field are executed. Normally this specification should be used to control only transaction unique logic. The Auditor will execute the audit/logic operations defined under Phase 0. If an error is encountered, the audited field is marked in error and the error message code, associated with the operation, is assigned to the field. The audited field does not have to be changed for this phase to be processed, although the field can be marked as changed through audit/logic operations. Controlling the changed flag can be used to force an audit dynamically in the Field Audit/Logic routine depending upon field values checked in the Automatic Field Assignment routine.

Field Audit/Logic Phase

After the Automatic Field Assignment phase is complete for all fields described in the Input Transaction Rule, the Auditor begins processing the Field Audit/Logic phase. The phase is specified as P1 in the Audit Language coding. All SLRs currently loaded in the SPA area are checked to determine if any fields require field audit processing. For a field to be processed within this routine, it must be marked as changed or be a KEY field. If a field is modified during Primary Key Audit, Secondary Key Audit or Preaudit, it is <u>not</u> marked as changed.

A field can be marked as changed in one of four ways.

- First, if the field has been modified through transaction input, it is marked as changed at the time it is updated in the SPA area. For example, a field is modified on a screen and entered. The input is mapped to the appropriate segment field in the SPA and the field entry in the Segment Layout Rule is marked as changed.

- The second way is in the Automatic Field Assignment phase (P0), either through specifying FAUDIT=Y or specifying AFA=Y and creating audit/logic operations that set the changed flag on.

- A third way is through a MAPPER call (into the SPA area) issued from the Auditor, a Special Processing routine or a user written audit exit.

- The final way is through a previous standard audit operation which modified the contents of a field.

If the changed flag is on or the field is a key, the Auditor verifies that:

- A field designated as ALPHA contains only alphabetic characters or blanks.

- A field designated as NUMERIC contains only numeric characters.

- A field designated as DATE contains a valid month (1-12) and a valid day for the month (i.e. Jun 1-30, Dec 1-31), or contains all zeros. If the field contains all blanks, the Auditor converts the blanks to zeros prior to date verification.

The field is marked in error if any condition is invalid.

The Auditor now determines if this field should be audited in the P1 phase. A Rules Generator specification of AUDIT=Y will mark the field in the Segment Layout Rule as requiring audit/logic processing in this phase. The Auditor will access the phase P1 Audit operations if AUDIT=Y and the changed flag is on, or AUDIT=Y and the field is a key. This phase should be used for audits which are used to edit the contents of a field each time it is modified.

Message Sending Phase

After all fields currently in the SPA area are processed through the Field Audit/Logic phase, the Auditor begins the final phase, Message Sending. This phase is specified as P2 in the Audit Language coding. This phase is only executed if processing in the previous two phases detects no errors or warning messages. If any P0 or P1 phase auditing generates an error or warning, the Message Sending phase is bypassed. Error or warning messages generated from the two previous phases are routed to a screen or transaction register, while messages generated from the Message Sending phase are sent to a Project/Group(s) and/or USERID(s) mail box. All SLRs currently loaded in the SPA area are checked to determine if any fields require message sending processing. A Rules Generator specification of MSG=Y will mark a field in the Segment Layout Rule as requiring message sending. A field is a candidate for message sending if:

- MSG = Y and the field is changed

- MSG = Y and the field is a key

If a field is a candidate, the audit/logic operations are executed. Since the Message Sending phase is only invoked after all previous audit/logic processing is completed correctly, it may be useful to use this phase for final field calculations and updates, prior to update of the data base. Informational messages can also be triggered for Project/Group or USERID destinations. If an informational message is to be sent, the operation specifies a Message Group Code which points to rules in the Message Data Base. These rules specify the message content and specific destination(s) to which it is sent. If any message sending is required the audited field flagged, and the Message Group Code is associated with the field.

Upon completion of the final phase, the Auditor returns to the Transaction Driver or Special Processing Routine that initiated the call.

The data base segments will not be updated by the transaction driver if errors occur during the auditing process. If no errors have been found, the transaction driver determines if any messages should be sent to a USERID or PROJECT/GROUP. If so, the messages are created and distributed as specified by the Message Rules in the Message data Base. If the return is to a Special Processing Routine, the return code (SPARTNCD) should be checked and appropriate action taken as described in the Special Processing section of Chapter 6, "Special Processing and User Exits."

ERROR ACTIONS

If the return is to the transaction driver, the following action(s) will be taken if an error is designated:

Transaction Processing	Audit Call	Action on Error
Conversational	Primary Key Secondary Key Preaudit	Error messages are displayed on the screen. Return is to the Primary Key Selection Screen.
Conversational	Standard Audit	Data screen is redisplayed with fields in error highlighted. Either the message 'ENTER 'E' TO DISPLAY ERROR MESSAGES' or the error message is displayed. Errors can be corrected or error messages can be displayed.
Nonconversational	Primary Key Preaudit	Error messages are displayed on the 3270 terminal or the 3284-6 printer.
	Secondary Key	Does not apply.
Nonconversational	Standard Audit	Error messages are displayed on the 3270 terminal or the 3284-6 printer.
Batch	Primary Key Preaudit	Error messages are printed on the SYSOUT data set.
	Secondary Key	Does not APPLY.
Batch	Standard Audit	Error messages are printed on the SYSOUT data set.

AUDIT/LOGIC RULES

Audit/logic processing is controlled by a combination of static and dynamic rules. The static rules (Input Transaction and Segment Layout) are used to determine when audit/logic processing should occur and the dynamic rules (Audit Data Base) specify the functions to invoke.

Since the Input Transaction Rule is associated with a transaction, the audit/logic requirements for a field can be varied by transaction. For example, one transaction requires that a field be audited each time it is retrieved, while another transaction does not require any auditing on the field. In the first case, AFA=Y is specified when the transaction is defined. In the second case AFA=N (or default) is specified for the field.

Figure 5-1 shows the Rules Generator operands which pertain to Auditing. The table shows the relationship between these operands, the Auditor calls, and the phases. A blank in the table denotes that the operand is ignored for that call. A number signifies a possible Audit access and references a following note.

RULE GENERATOR OPERAND	RULE TYPE	AUDITOR CALL				PHASE ACCESSED
		PRI-KEY	SEC-KEY	PREAUDIT	STANDARD	
	#	KEY-P0	KEY-P1	PRELIM	PROCESS	
KAUDIT = P	ITR	1				P0
KAUDIT = S	ITR		2			P1
PAUDIT = Y	ITR			3,6		P1
AFA = Y	ITR			4,6	10,6	P0
AUDIT = Y	SLR				3,7	P1
MSG = Y	SLR			5,6	11,7	P2
FAUDIT = Y	ITR				8	—
REQ = Y	ITR				9	—

Figure 5-1. Auditor operands and Auditor calls

\# Denotes Audit language definition of the call/phase (see Chapter 3, "Audit Language").
1 Valid only for key fields in a loadable DBPATH segment (KEY=Y).
2 Valid only for key or related fields in a loadable DBPATH segment (KEY=Y, RELATED=Y).
3 Causes auditing in the FAL leg.
4 In conjunction with PAUDIT=Y causes auditing in the P0 phase.
5 In conjunction with PAUDIT=Y causes auditing in the P2 phase.
6 Does not require the change flag on.
7 Also requires the change flag on or a key field (KEY=Y).
8 Turns on the change flag prior to AUDIT=Y or MSG=Y processing.
9 Requires the field have a non-initialized value.
10 Causes auditing in the AFA leg.
11 Causes auditing in the MS leg.

Audit rules are defined in two ways. The first way, described in the following sections, defines each rule as a specific operation denoted by a two-character alphanumeric code. The second way, described in Chapter 3, "Audit Language" uses an Audit Language to provide a high-level coding language to define the rules. The final output of either method will create the descriptor segments described in the following sections.

NAMING AUDIT/LOGIC RULES

Each audit operation is keyed with a 16-byte identifier. Depending upon the type of audit routine created; the key is constructed in one of five formats.

Format 1: Used to specify Audit records for Preaudit, Key Audit, Standard Audit and Key Audit (KANAME=ALT) calls and contains an eight-byte Audit group code plus an eight-byte fully qualified field name. The Audit group code is formatted as follows:

ssssyyyy where

ssss = Application System Identification

yyyy = Audit group - Specified in the Rules Generator operand 'AGROUP'

Since Audit group code is associated with an Input Transaction Rule, a field can have different audit/logic processing dependent upon the Input Transaction Rule currently being processed.

The eight-byte field name consists of:

ssxxffff where

ss = Two high-order bytes of Application System ID

xx = ID of segment that contains the field

ffff = Field ID

This name matches the Segment Layout Rule field name created by Rules Generator input.

Format 2: Used to specify Primary and Secondary key audits and is constructed as follows:

KEYAUDITssxxffff

The first eight bytes are a constant 'KEYAUDIT' followed by an eight-byte field name. Key Audits for a field will reside within a different data base record than Preaudits or Standard Audits for the same field. As discussed earlier, Primary Key Audits use the P0 phase and Secondary Key Audits use the P1 phase.

Key Audits also can be constructed using format 1. The Auditor will retrieve Key Audit operations using format 1 if KANAME=ALT has been specified on the Input Transaction Rule. If KANAME=ALT is not specified format 2 will be used. Refer to KANAME parameter in Chapter 2, "Rules Generator."

Format 3: Used to specify "common audits". An installation may have fields of the same name used in many segments and applications. An example might be a date field called DATE. The audit/logic processing for this field is constant for all occurrences. The user can designate the field name as a "common field" through Rules Generator input. This is done by specifying CAUDIT=Y each time the field is described to the Rules Generator. This will mark the field entry in the Segment Layout Rule as a Common field. When a field name is specified as common, only one set of audit/logic descriptors need be created and maintained for all of its occurrences. The key for a common field has the following format:

COMMON000000ffff

The first twelve bytes are constant (COMMON000000) and bytes 13-16 contain the Field ID. Upon detection of a common field, the Auditor will construct the root key as shown above.

Format 4: Used to specify audit subroutines. As discussed later, an audit operation is available which allows branching to an audit subroutine from any other audit routine. In this operation, the user specifies the 16-byte root key of the subroutine. Any meaningful 16-byte definition may be used for this key. For example:

```
SUBROUTINE-#0001
       or
0000000000000001
```

would both represent a valid key. If audit load modules are used, the first eight characters of this format are restricted to alphabetic or numeric characters and the first character must be alphabetic.

Format 5: Used to specify an audit table as described under table operations. Any meaningful 16-byte definition may be used for the key. If audit load modules are used, the first eight characters of this format are restricted to alphabetic or numeric characters and the first character must be alphabetic.

AUDIT DESCRIPTORS

The audit/logic rules are divided into Operation Descriptors and Data Descriptors. The Operation Descriptors specify the audit/logic operation to be performed and the next Descriptor to be accessed for the true or false condition. The Data Descriptors contain constant parameters, values or masks to be used in an operation. Some Operation Descriptors require one or more Data Descriptors to function correctly.

Operation Descriptors

Operation Descriptors are like logical instructions which, when combined, will form an audit/logic routine to process a field. Each Operation Descriptor invokes a specific function that is included as a subroutine in the Auditor. Each subroutine is identified by a descriptor code. For example, if field AAAA must be equal to field BBBB, a descriptor code of 30 would specify a subroutine to compare the audited field (AAAA) to a related field (BBBB). Depending upon the outcome of an operation, either a true or false condition is set. The logic flow is controlled through the testing and branching on these conditions. Messages also can be associated with Operation Descriptors. In the example above, if field AAAA did not equal BBBB, the application might require that an error message be displayed to the terminal user. A message number, which points to a message in the Message Data Base, is included in the Operation Descriptor. The following is a description of the fields in an Operation Descriptor.

SEQUENCE NUMBER	DESCRIPTOR CODE	RELATED FIELD	NEXT TRUE	NEXT FALSE	MESSAGE NUMBER/ CODE

Sequence Number - A two-character primary key.

Descriptor Code - A two-character descriptor code that indicates the audit operation to be performed.

Related Field Name - An eight-character field or function name

Next True Sequence - The two-character sequence number of the descriptor segment to be interpreted next if the results of the audit are true.

Next False Sequence - The two-character sequence number of the descriptor segment to be interpreted next if the results of the audit are false.

Message Number/Code - If audit is not successful, place the four-digit message number in the proper field in the Segment Layout Rule. For the Message Sending phase, the three final digits of this code are used to match the two-digit message group code and the one-digit format code that points to the appropriate message routing segment in the Message Data Base.
(See Chapter 4, "Dynamic Rules Data Bases.")

Figure 5-2. Operation Descriptor Format

The sequence number field is a two-character, alphanumeric sequence code. This code is the primary key for the descriptor record. The first physical record is accessed as the beginning operation. From that point on, the logic flow of the audit function is controlled by the next true or next false sequence key found in the appropriate fields (i.e., fields four and five) of the descriptor segment, or by an explicit branch specified by Operation Descriptor code 20.

The descriptor code field contains the code that defines the operation to be performed. This descriptor code corresponds to a function built into the Auditor or into a user-written routine.

Figure 5-3 contains a complete list of the descriptor codes and the functions they represent.

Code	Function
01	No operation, always returns a true value.
02	True if audited field value is > related field value.
03	True if audited field value is < related field value.
04*	True if audited field value is > value in Data Descriptor.
05*	True if audited field value is < value in Data Descriptor.
06*	True if related field value is > value in Data Descriptor.
07*	True if related field value is < value in Data Descriptor.
08*	Send a Secondary transaction. The OUTPUT SEGMENT name (8 characters) is contained in the Data Descriptor segment.
09*	Assign value to related field from Data Descriptor segment.
10	Assign value to audited field from related field.
11	Assign value to related field from audited field.
12*	Assign value to audited field from Data Descriptor segment.
13	Assign current date to any field. (Field must be 1-6 char with a type of ALPHANUM, DATE, DEC, or NUM - assigned format is YYMMDD).
14	Assign current time to any field. (Field must be 1-8 char with a type of ALPHANUM, NUM, or DEC - assigned format is HHMMSShh).
15	Mark any field as changed. (Used in the AFA descriptors to force an audit.)
16	True if this is a PREAUDIT pass. This descriptor should be used if a field has different audit/logic processing in a Preaudit and Standard Audit pass.
17	True if any bit field is on. (See Segment Layout Rule for bit specification.)
18	Set any bit field on. (See Segment Layout Rule for bit specification.)
19	Set any bit field off. (See Segment Layout Rule for bit specification.)
20*	Transfer control to the Operation Descriptor Segment specified in the Data Descriptor segment.
21*	True if audited field value is within the range contained in Data Descriptor segments. One pair of values per segment.
22*	True if audited field value matches one of the values contained in one Data Descriptor segment.
23*	True if audited field value matches one of the values contained in one or more Data Descriptor segments.
24	True if any field value is all numeric. (Field type must be ALPHANUM, NUM or DEC).
25	True if any field value is all alphabetic.
26	Set switch ON.
27	Set switch OFF.
28	True if switch ON.
29	Set a warning message on any field and continue with the audit.
30	True if audited field value matches related field value.
31*	True if related field value is in range contained in Data Descriptor segments. One pair of values per segment.
32*	True if related field value matches one of the values contained in one Data Descriptor segment.
33*	True if related field value matches one of the values contained in one or more Data Descriptor segments.

Figure 5-3 (Part 1 of 4). Audit Descriptor Codes and Their Functions

Code	Function
34	True if related field value is all numeric. (Field type must be ALPHANUM, NUM or DEC).
35	True if related field value is all alphabetic.
36*	Perform DL/I call specified in the Data Descriptor segment.
37*	True if status from previous DL/I operation matches one of the values in the Data Descriptor segment.
38	Mark related field changed.
39	True if related field was changed.
40	True if any field value contains no embedded blanks.
41*	False if any field contains any printable character listed in one Data Descriptor segment, (X, Y, Z, 1, 2, 3, etc.).
42	True if any field contains only alphanumeric data (no blanks).
43*	True if the non-$ characters in one dollar sign Data Descriptor matches those in any field, ($$AB$, $$DC$, etc.).
44	True if any field contains only alphabetic characters or blanks.
45	True if any field contains only numeric data or blanks.
46	True if any field contains alphabetic, numeric, or blank characters.
47	True if related field was marked in error.
48*	Round accumulator counter to significant digits specified in the Data Descriptor segment.
49	Reset accumulation counter to zero.
50	Add audited field value to accumulation counter.
51	Add related field value to accumulation counter.
52*	Add Data Descriptor value to accumulation counter.
53	Subtract audited field value from accumulation counter.
54	Subtract related field value from accumulation counter.
55*	Subtract Data Descriptor value from accumulation counter.
56	Multiply audited field value by accumulation counter.
57	Multiply related field value by accumulation counter.
58*	Multiply Data Descriptor value by accumulation counter.
59	Divide audited field value into accumulation counter.
60	Divide related field value into accumulation counter.
61*	Divide Data Descriptor value into accumulation counter.
62	Move accumulation counter to audited field.
63	Move accumulation counter to related field.
64@	Set any field to non-modifiable mode.
65@	Set any field to non-display mode.
66*	True if active Project/Group matches one of the values in one or more Data Descriptor segments.
67*	True if active transaction mode matches one of the values in one or more Data Description segments.
68*	True if active USERID matches one of the values in one or more Data Descriptor segments.
69@	Set any field to highlight mode.
A0	True if audited field equals accumulation counter.
A1	True if related field equals accumulation counter.
A2*	True if Data Descriptor value equals accumulation counter.
A3	True if audited field > accumulation counter.
A4	True if related field > accumulation counter.
A5*	True if Data Descriptor value > accumulation counter.
A6	True if audited field < accumulation counter.
A7	True if related field < accumulation counter.
A8*	True if Data Descriptor value < accumulation counter.

Figure 5-3 (Part 2 of 4). Audit Descriptor Codes and Their Functions

Code	Function
A9*	Set screen attribute.
B0*	Decode audited field to related field.
B1*	Decode related field to audited field.
B2*	Encode audited field to related field.
B3*	Encode related field to audited field.
B4*	True if audited field is a table argument.
B5*	True if audited field is a table value.
B6*	True if related field is a table argument.
B7*	True if related field is a table value.
B8*	True if active TRXID matches one of the values in one or more Data Descriptor segments.
B9*	True if active LTERM - same as above.
C0*	Move related field1 to related field2.
C1*	True if related field1 =1 related field2.
C2*	True if related field1 > related field2.
C3*	True if related field1 < related field2.
C4*	Combination Add (A+B=1C)
C5*	Combination Subtract (A-B=1C)
C6*	Combination Multiply (AxB=1C)
C7*	Combination Divide (A/B=1C)
C8*	Issue ROLLCALL and DISPLAYE.
C9*	Initialize a segment area.
D0*	True if segment has been retrieved.
D1*	Set SPAERMSG.
D2	Move a related field to SPAKEYID.
D3	Move SPAKEYID to a related field.
D4*	Change transaction ID.
D5*	Change transaction mode.
D6	Prohibit segment display. (Secondary Key Selection only).
D7	Assign Project Group to any field.
D8	Assign transaction mode to any field.
D9	Assign USERID to any field.
E0	Assign TRXID to any field.
E1	Assign LTERM to any field.
E2*	Perform a SETFLAG operation.
E3	Prohibit segment display and terminate Secondary Key Selection retrieval.
E4*	Set twin segment IDs.
E5*	DOTWIN occurrences x to y.
E6	ENDTWIN occurrence processing.
E7	Assign SYSID to any field.
E8*	True if SYSID matches one of the values in one or more Data Descriptor segments.
E9*	Change PCBNO for one or more segments.
F0*	Direct a secondary transaction to a Multiple System Couple link.
F1	Set any field in error and continue processing.
F2	True if this is a Key Audit pass. This descriptor should be used if a field has Key Audit logic and either Preaudit or Standard Audit logic under the same root key.
F3	Set color attribute.
F4	Set extended highlight attribute.
F5	Set first field in an array.
F6#*	Check SPAULANG field against data descriptor value.
F7#*	Set field outlining attribute.
F8#	True if field type is DBCS or if field type is MIXED and field contains DBCS data.
G0#*	Concatenate 2 fields.
G1#*	Move from a substring.
G2#*	Move to a substring.
G3#	Set SPASQLKS in the SPA.
G4#	Set SPAWHERE in the SPA.
G5#	Test a column for NULL.
G6#	Test a column for truncate.

Figure 5-3 (Part 3 of 4). Audit Descriptor Codes and Their Functions

Code	Function
G7#	Set a column to NULL.
G8#*	Execute a SQL call.
G9#*	Check SPASQLCD or SPAWARNn for equal value.
H0#*	Check SPASQLCD for greater than value.
H1#*	Check SPASQLCD for less than value.
H2*$	Call MAPPER.
H3*$	Call COPYSEG.
H4$	True if segment changed flag is on.
H5$	True if there is any previous error in this Auditor call.
H6$	Unconditional exit from the Auditor.
70-99 & W0-Z9	These descriptor codes are reserved for user-written audit routines. For more detailed information, see the subsection "User Audit Exit Processing" in Chapter 6, "Special Processing and User Exits."
*	Requires at least one Data Descriptor segment to perform the function.
@	Maintained for compatibility - Replaced by operation A9.
#	New with Version 2.
$	New with Version 2 Release 2.

Figure 5-3 (Part 4 of 4). Audit Descriptor Codes and Their Functions

Note: Audited field refers to the field currently being audited. Related field refers to a field specified in the Related Field Name section of the Operation Descriptor. Any Field denotes that either the audited or a related field can be used. If the Related Field Name is blank, the operation defaults to the audited field.

Each descriptor code has a corresponding subroutine in the Auditor. These subroutines are invoked by specifying the appropriate descriptor code. The subroutines perform the appropriate functions and set a status code to either a true or false condition. Subroutines that move or compute data values will always return a true condition, while the remainder return true or false depending upon the outcome of the operation.

If an error occurs within the subroutine (i.e., Related field not found, Data Descriptor Segment not found, incorrect data, etc.), the field is marked in error, and an internal Auditor message is assigned for display. Audit/Logic processing terminates on the field, and continues on the next field in sequence.

The related name field contains the name of another field or keyword. Some descriptors are checked for information contained in related fields. When an audit is being performed on a field, that audit can be successful or unsuccessful based on values found in other fields. The eight character related field name indicates the related field to be checked. The field name is represented as ssxxffff as described in the previous section ("Audit/Logic Rules"). Any field in a segment currently residing in the SPA area can be designated as a related field. If a descriptor function requires the use of a related field, the segment table is scanned to determine the location and attributes of the related field. If the related field is not found, the Auditor returns a message code indicating that the related field could not be found for audit and marks the audited field in error.

The next true and next false sequence fields normally contain the sequence number of the next audit descriptor to be interpreted. That descriptor describes the next function to be performed. If the status code from the subroutine is true, the next true sequence is used. If false, the next false sequence is used.

There are two special values for the next true or next false sequences. These are a next sequence of 00 or a next sequence of blanks. A next sequence of 00, when selected, signals that an audit has completed successfully and no message code/number is to be returned for the field audited. Blanks in a next sequence field indicate that the user is to be informed of an error or critical condition through the contents of

the message number/code field. Either condition causes the Auditor to cease audit/logic operations on the field and proceed to the next field.

The message number/code field contains one of two formats. (1) A four-digit message number (nnnn) is placed in the Segment Layout Rule if an audit is unsuccessful. This number, preceded by the System ID (ssssnnnn), forms the key of a Message Generation segment in the Message Data Base. (2) If this field is used in the Message Sending phase, it contains a four-digit number used to select the correct Automatic Message Routing segment in the Message Data Base. This number is formatted in the message code field as follows:

```
        0Cnn

where   0 = constant 0
        C = code (0-3) which specifies the format
            of the Automatic Message Routing segment key.
            The formats are:

            0 = spgmxxnn          ss = First two characters
                                       of System ID
            1 = ss######          pg = Project/Group
            2 = ss##pg##           m = Processing Mode
            3 = ss####nn          xx = Segment ID
                                   # = (#'s are Constant)

        nn = Message group code

        Depending upon the C and nn values, the system selects the
        appropriate Automatic Message Routing segment. This supplies
        information on messages to send, and the Project/Group and/or
        USERID distribution(s).
```

The message number or code is placed in the communications area of the Segment Layout Rule for the field being audited. During the Message Sending phase, the message flag is turned on in the flag area of the field's Segment Layout Rule. If the field is in error, the error flag is turned on in the flag area of the field's Segment Layout Rule. The field in error will be shown on the screen in high intensity and a message will be displayed to point the terminal operator to the error message. If the terminal operator wishes further information about the error, he enters "E" to display the error message.

Refer to "Updating the Audit Data Base" in this chapter for information pertaining to creation and maintenance of Data Descriptor segments.

Data Descriptors

Data Descriptors are associated with an Operation Descriptor. They contain the constants and literals used by an audit/logic operation to check for values, ranges, or tables. They also may contain additional parameters for the function being performed. Not all audits require Data Descriptor. An example of an audit/logic operation that does not, is the assignment of the current date to a field.

The audit descriptor codes that require one or more Data Descriptors are shown with an * in Figure 5-3. The data contained in the Data Descriptors are either values that are used for compares, ranges of values within which the input data must fall, values to be assigned to an audited or related field, computational values, table names, field names or function operands. If a required Data Descriptor cannot be found, the field is marked in error and a message is displayed with the cause.

Data Descriptor records are stored in a DL/I data base in the following format: If a DB2 table is used the format is slightly different.

```
    xxxx(value1,value2,...,valuen)
```

where xxxx is a four-digit sequence number.

The first sequence number must always be 0001. The remainder of the segment contains values separated by commas and enclosed within

parentheses. Commas and parentheses cannot be used as values within segments.

An example of Data Descriptor segment for descriptor code 12 is shown in Example 1.

```
Example 1:  0001(12345)
```

Descriptor code 12 specifies that the value in the Data Descriptor segment is to be assigned to the field being audited. If the field being audited is not specified as EBCDIC, the contents of the Data Descriptor segment are converted to the characteristics specified for the field.

There are three formats in which individual values can appear in Data Descriptor segments. If all individual values can be contained within a single record, they can appear in one segment within parentheses and separated by commas. If a single segment cannot contain all values, then multiple segments must be used. Each segment contains one or more unique values within parentheses. The following examples show how these formats are used.

```
Example 2:  0001(A,B,C)
```

Example 2 shows a Data Descriptor segment with three values A, B, and C. This form would be used with a compare operation such as descriptor code 22.

```
Example 3:  0001(A)
            0002(B,C)
```

Example 3 shows the same three values in two Data Descriptor segments. This form would be used with a compare operation such as descriptor code 23.

A final way to store the data is in spanned format, where up to three Data Descriptor segments are concatenated to form a single field up to 70 bytes. The following example shows this technique comparing a 25-byte audited field against two values.

```
Example 3.1:    0001(TWENTY-FIVE BYTE FIELD
                0002#1,TWENTY-FIVE BYTE FIEL
                0003D#2)
```

Parentheses enclose the data string. The Auditor reads all three segments and treats the combined field as:

```
(TWENTY-FIVE BYTE FIELD#1,TWENTY-FIVE BYTE FIELD#2)
```

This technique may be used for audit/logic operations on a field greater than 22 bytes. If more than three segments are read before a ')' is encountered, an error condition occurs and the field is marked in error and auditing is terminated.

Data Descriptor segments that contain value ranges have one set of inclusive ranges contained within parentheses per segment or concatenated segment. The high and low values within a range are separated by a comma. If several groups of ranges are required for an audit, then as many segments are required as there are groups. Example 4 illustrates the use of Data Descriptor segments for specifying ranges.

```
Example 4:  0001(1000,5000)
            0002(7000,9000)
```

This example shows two Data Descriptor segments that contain the value ranges against which the value in a field will be compared. This form would be used for range check operations such as descriptor codes 21 and 31.

The Data Descriptor Segment for a compare against a mask (descriptor code 43) is used in the following manner:

```
Example 5:  0001($AB$,A$$1)
```

For a Data Descriptor segment specified as shown in the example, true would be returned for fields containing such values as AABB, AA11, and ZABX. FALSE would be returned for fields containing values such as ABCD, and 4321.

Refer to "Updating the Audit Data Base" in this chapter for information pertaining to creation and maintenance of Data Descriptor segments.

COMPARE OPERATIONS

Subroutines are provided to compare field values, constants, or field content (NUM, ALPHA, etc.). The audited field may be compared against a related field, an accumulation counter, an auxiliary counter or constant value(s) contained within Data Descriptor segments. A related field may be compared against another related field, an accumulation counter, an auxiliary counter or constant value(s) in Data Descriptor segments. Both types of fields can be checked for numeric or alphabetic content. Compare operations dealing with only the audited field will be blank in the Related Field Name. If one related field is required, it is specified in the Related Field Name. If two related fields are required (Codes C1, C2, C3), the Data Descriptor will specify both field names in the following manner:

```
0001(relatedfield1,relatedfield2)
```

Any compare operation using constant data will require one or more Data Descriptors.

Each compare routine will return a true or false indication depending upon the outcome. Fields with an attribute of binary, decimal, or packed decimal will be converted to a signed packed decimal number prior to the compare. Other field types will use a logical compare with either the length of the audited field, or the Data Description value length, if one is used. A field with a numeric attribute may be compared either way depending upon the attribute of the other field or compare value.

Figure 5-4 defines the type and length of compares performed between field types. As shown, compares between certain field types are not valid. If a compare must be made between two incompatible field types, one field should be moved to a common type before the compare can be performed.

Field Type	a	n	c	z	d	p	i	b	h	dd	v	f	db	m
alphabetic	1	1	1	1	1	x	x	xa	1	1	1a	x	x	1
numeric	1	1	1	2	1	2	2	xa	1	1	1a	x	x	1
alphanumeric	1	1	1	1	1	x	x	xa	1	1	1a	x	x	1
decimal	1	2	1	2	1	2	2	xa	1	2	1a	x	x	1b
date	1	1	1	1	1	x	x	xa	1	1	1a	x	x	1b
packed decimal	x	2	x	2	x	2	2	xa	x	2	x	x	x	x
binary	x	2	x	2	x	2	2	xa	x	2	x	x	x	x
bit	xa	xa	xa	xa	xa	xa	xa	xa	xa	xa	xa	x	x	x
hexadecimal	1	1	1	x	1	x	x	xa	1	3	1a	x	x	1b
data descriptor constant	1	1	1	2	1	2	2	xa	3	z	1a	xa	1	1b
varchar	1a	1a	1a	1a	1a	x	x	xa	1a	1a	x	x	x	1b
float	x	x	x	x	x	x	x	x	x	x	x	x	x	x
DBCS	x	x	x	x	x	x	x	x	x	x	x	x	1	1b
mixed	1	1	1	1b	1b	x	x	x	1b	1b	1b	x	1b	1b

Legend:

1 = Logical compare using the length of the shortest field or data descriptor value.
1a = Same as 1, except that a zero length varchar field will be compared as all blanks with its maximum length
1b = Depends on pure/mixed contents - see DBCS Section.
2 = Packed decimal compare. Both operands are in eight character packed decimal fields.
3 = Logical compare using the length of the hexadecimal field.
x = Not a valid compare. Results are unpredictable. The field(s) should be moved to a common data type before the compare.
xa = Bit fields can only be tested on or off by operation code 17.
z = Cannot be compared.

Figure 5-4. Matrix of Auditor Compares

Decimal alignment is performed only on compares against the accumulation counter. On all other compares, decimal alignment is ignored and the values are treated as whole numbers.

Another category of comparison allows the audit/logic routine to vary depending upon system status information. For example, if the audit/logic criteria for a field is different between add mode and update mode, then Operation Descriptor code 67 is used to determine which mode is being processed. Using codes 66 or 68, a routine also can be varied on the Project/Group or USERID currently signed on. Codes B8 and B9 uses Transaction ID or logical terminal for the compare. Code E8 compares on the SYSID currently in control. This code might be used in a subroutine called from various major application systems. Each of these compares requires one or more Data Descriptor segments which contains a list of one or more values. If any value in the list matches the corresponding system data, the compare is true.

If Preaudit and Standard Audit passes are both used on a field, then operator code 16 normally will be required, since the same audit/logic routine is used on each pass. For example, field AAAA is checked on a Preaudit pass to determine if it should be displayed for update. If so, after the update has occurred, a Standard Audit pass is used to verify the new field content. The first Operation Descriptor in the audit/logic routine should have operation code 16. The next true sequence would point to the Preaudit descriptor(s). The next false sequence would point to the Standard Audit descriptor(s).

Examples of compare operations: The following set of Descriptors demonstrate compare operations against the following fields:

```
Audited Field   SACDAAAA    six-character, ALPHANUM
Related Field   SACBBBBB    six-character, two decimal, PD
Related Field   SACDCCCC    eight-character, ALPHANUM

SAMPAUDTSACDAAAA                Root segment (16-byte key)
0122        02 1111            If audited field (1-4)=ABCD go
0001(ABCD)                        to 02; else error message 1111.
0222        03 2222            If audited field (1-6)=ABCDEF go
0001(ABCDEF)                      to 03; else error message 2222.
0306SACBBBBB04 3333            If related field > 300.50 go
0001(300.50)                      to 04; else error message 3333.
0430SACDCCCC05 4444            If audited field=related field
                                  (1-6) go to 05; else error
                                  message 4444.
0568        0006                If current USERID=one of 5 listed,
0001(111111,222222,333333,44      terminate the audit for this
00024444,555555)                  field; else go to 06.
0667          005555         If mode is not Remove (3)
0001(3)                           terminate the audit; else
                                  error message 5555.
```

DATA MOVE AND FIELD CONTROL OPERATIONS

Values may be assigned to an audited field from either a related field or from a Data Descriptor constant. Values may also be assigned to a related field from the audited field, another related field or a Data Descriptor constant. Data conversion occurs if the receiving field and source data have different attributes. Refer to Appendix B, "Data Conversions and Mappings" for detailed information on conversion and sign results. The descriptor formats previously described under Compare Operations, for audited field only, one related field and two related fields also apply to moves.

Descriptor codes 62 and 63 can be used when arithmetic operations are being performed. These codes allow the result of arithmetic operations to be moved from the accumulation counter to either the audited field (62) or a related field (63). The field receiving the answer must have a numeric, binary, decimal, or packed decimal attribute.

All move operations to PD, DEC, or NUM fields will use the SIGN attribute of the receiving field to determine the sign configuration as follows:

Type	Sign= Attribute	Result Sign +	Result Sign -
PD or DEC	Y	'C'	'D'
PD or DEC	N	'F'	'F'
NUM	-	'F'	'F'

If SIGN=N or the field is Numeric, the result sign will always be 'F'. Fields which may contain a minus result must be defined as SIGN=Y (default).

The source field of a move operation from a data descriptor or alphanumeric field to a DATE field should be in the format specified by the DATEFMT parameter at installation time. The default format is MM/DD/YY. The values will be rearranged to the format YYMMDD in the target field.

The following system fields can be moved to an audited or a related field:

System Field	Code	Description of Receiving Field
Current Date	13	(one- to six-character field. Type = Alphanumeric, Decimal, Date or Numeric. FORMAT-YYMMDD)
Current Time	14	(one- to eight-character field. Type = Alphanumeric, Decimal or Numeric. or Numeric. Format = HHMMSSHH)
Project/Group	D7	two-character field (Type = Alphanumeric,
Transaction Mode	D8	one-character field Numeric,
USERID	D9	six-character field Decimal, or
Transaction ID	E0	two-character field Alpha)
Logical Terminal	E1	eight-character field
SYSID	E7	four-character field

CONCATENATION AND SUBSTRINGING

This section describes three operations which support concatenation and substring notation.

1. Op Code G0.

   Concatenation--Concatenate two fields into one field. The target field is named in the related field portion of the operation descriptor. The two source fields are named in the data descriptor. The Auditor concatenates the two source fields into one field and stores it in the target field. If the data is VARCHAR, the halfword length is not included. If the target is VARCHAR, the length portion is changed as necessary. If concatenation is attempted on any data type other than ALPHA, ALPHANUM, NUM, or VARCHAR, error message number 134 is generated as a terminating error. Note that blanks are significant. All of each source field is used in the concatenation. The field length is obtained from the SLR definition or the first halfword of VARCHAR data. If a source field is NULL, the concatenation is not done, and the target field is set to NULL. If only the target is NULL, the operation is done and the target field is set to not NULL.

   The following example concatenates field STR2 with STR3 and stores the result of the concatenation into STR1:

```
01G0BAPSSTR102020000          BAPSSTR1 = CONCAT BAPSSTR2 BAPSSTR3
0001(BAPSSTR2,BAPSSTR3)        or
                              STR1 = CONCAT BAPSSTR2 BAPSSTR3
```

2. Op Code G1.

   Substrings--move from a substring. The target field is named in the related field portion of the operation descriptor. The source field is named in the first data descriptor. The starting position into the source field and the length of the string are contained in the second data descriptor either as two literals or as two related field names which contain the values. If the target is VARCHAR, the Auditor will update the length portion of that field according to the length of the string moved into it. The Auditor assumes that the High Level Audit Language insures that when constants are used for position and length are both greater than zero. If the starting position is greater than the length of that field, the Auditor will perform the move for a length of zero. The Auditor will not allow fields to overlap by substringing past the end of the source field or assigning into the field behind the target field. If the source field is NULL, the substringing is not done and the target field is set to NULL. If only the target is NULL, the operation is done and the target field is set to not NULL. If substringing is attempted on any data type other than ALPHA, ALPHANUM, NUM, or VARCHAR, error message number 134 is generated as a terminating error.

   The following example moves seven bytes from STR2 beginning with the third byte into STR1:

```
01G1BAPSSTR102020000      BAPSSTR1 = SUBSTR BAPSSTR2 3 : 7
0001(BAPSSTR2)             or
0002(3,7)                 STR1 = SUBSTR BAPSSTR2 3 : 7
```

   The following example is similar except that the starting position and the length of the substring are stored in STR3 and STR4, respectively:

```
01G1BAPSSTR102020000      BAPSSTR1 = SUBSTR BAPSSTR2 BAPSSTR3 : BAPSSTR4
0001(BAPSSTR2)             or
0002(BAPSSTR3,BAPSSTR4)   STR1 = SUBSTR BAPSSTR2 BAPSSTR3 : BAPSSTR4
```

3. Op Code G2.

   Substrings--move to a substring. The target field is named in the related field portion of the operation descriptor. The source field is named in the first data descriptor. The starting position in the target field and the length of the string are contained in the second data descriptor either as two literals or as two related

field names which contain the values. If the target is VARCHAR, the Auditor will not change its length since the intent is to modify only a part of that target field. The Auditor assumes that the High Level Audit Language insures that when constants are used for position and length, that they are both greater than zero. If the starting position is greater than the length of that field, the Auditor will not perform the move, and will treat the operation like a NOP. Also, if the length of the subscript is a constant zero, the HLAL will flag an error. If the length of the subscript is contained in a field which contains zero, the operation is a NOP. The Auditor will not allow fields to overlap by substringing past the end of the target field substring. If the length of the target substring is less than the length of the source field, truncation will occur at the last position of the substring. If the source field is NULL, the substringing is not done and the target field is set to NULL. If only the target field is NULL, this is a NOP and the target field remains NULL. When the position and length are contained in related fields, those related fields cannot be NULL, i.e. no check for NULL will be made. If substringing is attempted on any data type other than ALPHA, ALPHANUM, NUM, or VARCHAR, error message number 134 is generated as a terminating error.

The following example moves seven bytes from STR2 into STR1 starting at the third position of STR1:

```
01G2BAPSSTR102020000      SUBSTR BAPSSTR1 3 : 7 = BAPSSTR2
0001(BAPSSTR2)             or
0002(3,7)                 SUBSTR STR1 3 : 7 = BAPSSTR2
```

The following example is similar except that the starting position and length of the substring are stored in STR2 and STR3, respectively:

```
01G2BAPSSTR102020000      SUBSTR BAPSSTR1 BAPSSTR2 : BAPSSTR3 = BAPSSTR4
0001(BAPSSTR4)             or
0002(BAPSSTR2,BAPSSTR3)   SUBSTR STR1 BAPSSTR2 : BAPSSTR3 = BAPSSTR4
```

Note that concatenation and substringing are only supported one step at a time. Direct concatenation of substrings and directly moving substrings to other substrings is not supported. Also, note that the user will need to use caution when substringing subdefined data. Results will not necessarily be the same when substringing different definitions of the same data.

CALLING THE DATA MAPPER

Operation descriptor code H2 can be used to map data to or from an area in the SPA according to a mapping segment definition. The operation and data descriptors generated by the audit language are:

```
01H2relfield02020000
0001(id,0)

       OR

01H2relfield02020000
0001(id,1)

       OR

01H2        02020000
0001(id,id)
```

The Auditor performs this operation by calling MAPPER in MFC1ESPI. The operation and data descriptor information is used to format the call. Relfield is the name of a related field that is the beginning of a contiguous area in a segment. If a related field is specified, the id that is specified must be a mapping segment id.

If the map is from one id to another, one id must be a mapping segment and the other must be a segment id in the transaction that is not a mapping segment.

A 0 in the data descriptor specifies that the data is to be mapped from the areas specified in the mapping rule to the area specified by related field.

A 1 in the data descriptor specifies that the data is to be mapped from the area specified by related field to the areas specified in the mapping rule.

If two id's are specified, the Auditor will format the mapper call so that the data is mapped from the first id to the second.

Information about how MAPPER works can be found in Chapter 6, "Special Processing and User Exits." Information about how to specify a mapping segment definition can be found under the Segment Statement in Chapter 2, "Rules Generator." Information about the MAP statement can be found in Chapter 3, "Audit Language."

CALLING THE COPYSEG ROUTINE

Operation descriptor code H3 can be used to copy all or part of a segment to or from another segment. The operation and data descriptors generated by the audit language are:

```
01H3relfield02020000
0001(id,0)

       OR

01H3relfield02020000
0001(id,1)

       OR

01H3        02020000
0001(id,id)
```

The Auditor uses the descriptor information to format a call to the COPYSEG routine in MFC1ESPI. The id specified is a segment id in the transaction. If a related field is specified, it must be the name of a field that is the beginning of a contiguous area in a segment to be used by COPYSEG.

A 0 in the data descriptor specifies that the data is to be copied from the specified id to the area specified by the related field. A 1 in the data descriptor specifies that the data is to be copied from the area specified by the related field to the specified id.

If two id's are specified, the Auditor will format the COPYSEG call so that the data is copied from the first id to the second.

If a related field is specified, COPYSEG will use the length of the segment whose id is specified. If two segment id's are specified, COPYSEG will use the length of the shortest segment.

After a COPYSEG statement is executed, the target segment's change flag will be on. The field change flags in that segment will not be on. If the use of field change flags is required, the MAP statement (descriptor code H2) is recommended.

Information about how COPYSEG works can be found in Chapter 6, "Special Processing and User Exits." Information about the COPYSEG statement can be found in Chapter 3, "Audit Language."

SCREEN ATTRIBUTE OPERATIONS

Screen attributes are controlled through operation Code A9. The Operation and Data Descriptors associated with this operation have the following format:

```
01A9          0202
0001(fieldname,function,setting)

fieldname = eight-character name of field whose screen attribute
            is modified.
function  = four-character code specifying which attribute is to be set.
                 HILT = Highlight
                 PROT = Protect
                 NDSP = Non-Display
                 PMOD = Premodify
                 CURS = Cursor
setting   = one-character code specifying the condition of the attribute
            after the operation.
                 0 = OFF
                 1 = ON
```

The color and extended highlight attributes on an IBM 3279 Color Terminal can be modified through the following two operations.

```
01F3          0202              Set color
0001(fieldname,color)

fieldname = eight-character name of field whose color attribute
            is modified
    color = Name of color.  Valid names are:
            PINK,BLUE,GREEN,RED,WHITE,YELLOW,TURQUOISE


01F4          0202              Set extended highlight
0001(fieldname,xhilt)

fieldname = eight-character name of field whose extended highlight
            attribute is modified.
    xhilt = Name of highlight attribute.  Valid names are:
            UNDERSCORE - Underscore the field
               REVERSE - Set field to reverse video
                 BLINK - Cause field to blink
               DEFAULT - Reset highlight to no value
```

The set color and set extended highlight operations are only processed against Input Transaction rules which are generated for color screens. These rules have expanded entries to hold color and/or extended highlight values. If these operations are processed against a standard Input Transaction rule they are ignored. No error message is generated.

If the field is not displayed on the screen an error message is generated and auditing is terminated for the audited field. One attribute is modified per operation.

ARITHMETIC OPERATIONS

Single arithmetic operations are performed by using descriptor codes 50 through 61. Each routine performs an operation between a field, related field, or Data Descriptor value and an accumulation counter. For example, the computation (field AAAA = field BBBB + field CCCC) would be handled as follows:

```
0151SSXXBBBB02
0251SSXXCCCC03
0362          00
```

Descriptor 1 adds the related field (BBBB) to the accumulation counter. The counter previously was cleared by the Auditor. Descriptor 2 adds the related field (CCCC) to the accumulation counter. Descriptor 3 moves the accumulation counter to the audited field (AAAA).

Fields must have a numeric, binary, decimal, or packed decimal attribute to be processed by an arithmetic operator. If the field attributes vary within the operands of a calculation, they will be converted to a common base. The final answer will be converted to the attribute of the receiving field. The divide operators (59, 60, 61) will divide the accumulation counter by the value contained in an audited field, related field, or Data Descriptor and the resultant quotient will be placed in the accumulation counter.

The Auditor performs automatic decimal alignment on the accumulation counter value. Each arithmetic operation (50-61), arithmetic compare (A0-A8), or arithmetic move (62,63) uses decimal positions of both the current accumulation counter and the other operand in performing the required function. The accumulation counter, an eight-byte Packed Decimal field, accommodates up to 13 decimal positions. The following table shows various decimal alignments performed by the Auditor:

FIELD A	FIELD B	FIELD C	FIELD D	ACCUMULATION CTR
1.1	2.22	3.333	7.7777777	000000000000000
ADD				1.1
		ADD		4.433
	SUB			2.213
	MULT			4.91286
		DIV		1.47400
1.4	1.47	1.474	1.4740000	MOVE*
				1.47400
EQUAL	EQUAL	EQUAL	EQUAL	COMPARE**

* Move operations (62-63) use the receiving field decimal position and adjusts the accumulation counter value, by zero extension or truncation.
** Compare operations (A0-A8) use the decimal position of the field or accumulation counter, whichever is less.

Two additional operations control the value in the accumulation counter. The counter value may be reset to zero by using Operation Descriptor code 49. Decimal position is also reset to zero. Rounding to a significant decimal digit is accomplished with Operation Descriptor code 48. The associated Data Descriptor segment contains the significant digits desired after the round. For example, if the accumulation counter currently contains the value 22.4567 and is to be rounded to two decimal places, use the following audit/logic operation:

```
1548        0000
00001(2)
```

After the operation, the counter contains 22.46, and the decimal position is set at two.

In addition to single arithmetic operations, Codes C4 through C7 are used to perform a complete simple operation in one instruction. The previous example (field AAAA = field BBBB + field CCCC) would be handled as follows:

```
01C4SSXXAAAA02
0001(SSXXBBBB,SSXXCCCC)
```

Data Descriptor field2 (DD2) is added to Data Descriptor field1 (DD1) and the result is placed in the Related Field Name. The result, DD1 and DD2 fields, can be an audited field, related field, or auxiliary counter. The accumulation counter cannot be specified. Other operations are:

```
Subtract - Related Field Name = DD1 - DD2
Multiply - Related Field Name = DD1 X DD2
Divide   - Related Field Name = DD1 / DD2
```

Decimal alignment is computed from each operand as shown above. Rounding will not occur. Data Descriptor field2 (DD2) can also contain a constant value instead of a field name. For example, if the computation is (field AAAA = field BBBB/3.75), the descriptors will look as follows:

```
01C7SSXXAAAA02
0001(SSXXBBBB,3.75)
```

Constant values are not allowed in field1 or the Related Field Name.

Nine auxiliary counters (8 bytes Packed Decimal) are available to store intermediate results. These counters are specified COUNTER1...COUNTER9 and are referenced the same as a related field name. These counter designations are valid only for arithmetic and compare operations. For example, if the computation is (COUNTER1 = field BBBB COUNTER3), the descriptors will look as follows:

```
01C4COUNTER102
0001(SSXXBBBB,COUNTER3)
```

The counters are set to zero at the beginning of each Auditor call (Primary Key, Secondary Key, Preaudit or Standard Audit). They may be reset by using Code 49 and specifying the counter name in the Related Field Name. Compares are handled by specifying a related field compare, such as 32 or A1, and specifying the counter name in the Related Field Name. To move a constant or field value to a counter, an arithmetic operation such as (COUNTER1 = field BBBB + 0) <u>must</u> be used. To move a counter to a field, an arithmetic operation such as (field BBBB = COUNTER 1 + 0) <u>must</u> be used. Move operations between auxiliary counters and fields are not valid. The accumulation counter may be moved to an auxiliary counter with Operation Code 63 by specifying the auxiliary counter in the Related Field Name. Another counter (COUNTERF) will contain the remainder from each divide operation. However, COUNTERF will not necessarily contain a valid remainder unless both the divisor and the dividend have no decimal positions. This counter may be used in the same manner as COUNTER1-COUNTER9.

BRANCHING OPERATIONS

Branching within an audit/logic procedure is controlled by the Operation Descriptor key specified in the next true and/or false fields. Control is passed from one Operation Descriptor to another, similar to program branching. Control may be passed forward or backward.

A facility is also available through Operation Descriptor code 20 for branching from one set of Operation Descriptors to another set under a different key. This subroutine capability allows a single procedure to be accessible by any number of other audit/logic procedures.

Within the subroutine, operations which specify the Audited field point to the audited field of the calling routine. The key of the subroutine is any 16 characters. For example:

Descriptor code 20 can be used in three ways depending upon the format of the accompanying Data Descriptor:

```
     0120          0000                Operation Descriptor
1.   0001(SAMPAUDTROUTINE1)           Data Descriptor
2.   0001(SAMPAUDTROUTINE1,10)        Data Descriptor
3.   0001( )                          Data Descriptor
```

The first Data Descriptor demonstrates how to branch to another set of Operation Descriptors under a different key. In the example, control is passed to the first Operation Descriptor under the key SAMPAUDTROUTINE1. This is format 4 of the key described previously under AUDIT/LOGIC RULES. The phase in the subroutine must coincide with the phase from which the branch is made.

The second Data Descriptor controls branching to a specific Operation Descriptor within an audit/logic procedure. In the example, control is passed to Operation Descriptor with a sequence number of 10 under the key SAMPAUDTROUTINE1.

The third Data Descriptor controls the return to the calling routine at the Operation Descriptor causing the branch. This usage normally exists as the final descriptor in a subroutine used by other audit/logic procedures. Upon return to the calling routine, the next true field is checked for a ' ' or '00'. If present the next operation is accessed. If other than ' ' or 00, the value is used as the sequence number of the next operation. The subroutine assumes the characteristics of the calling routine. The audited field pointer of the calling routine is used in the subroutine. If an error is triggered in a subroutine the audited field of the calling routine is marked in error. In this way common auditing across many different fields can occur in a single subroutine. Subroutine calls may have five levels of nesting.

SECONDARY TRANSACTION OPERATIONS

Secondary transactions can be initiated through Operation Descriptor code 08. Secondary transactions are marked for sending at the end of the transaction by flagging the appropriate secondary transaction definition in the Input Transaction Rule. An Input Transaction Rule may have multiple secondary transaction definitions. Operation Descriptor code 08 allows the user to send a specific secondary transaction whose definition name matches the value supplied in the Data Descriptor segment. The secondary transaction definition name is constructed in the format ssORxx01 as follows:

ss - first two characters of the System ID
OR - constant
xx - two-character segid specified for the STX operand of the Rules Generator
01 - constant

For example:

A secondary transaction whose definition name is SAORAA01 must be sent whenever the value in field AAAA falls below the value in field BBBB.

```
0103SAIDBBBB0200
0208        0000
0001(SAORAA01)
```

Operation Descriptor 1 compares the audited field (AAAA) against the related field (BBBB). If the audited field is less than the related field, Operation Descriptor 2 is accessed. Otherwise, the audit is complete. Operation Descriptor 2 instructs the Auditor to flag the secondary transaction definition named SAORAA01. Upon return, the transaction driver will check all definitions in the Input Transaction Rule and will generate secondary transactions for any definitions that have been flagged.

Secondary transactions can be sent immediately by specifying 'IMMED' in the related field name of the Operation Descriptor.

```
0208IMMED   0000
0001(SAORAA01)
```

In the above example the secondary transaction defined as SAORAA01 will be sent during the execution of operation code 08. By using the immediate technique, a single definition can be used to send multiple secondary transactions as long as the format and destination are the same.

Under IMS/VS, secondary transactions can be routed to a Multiple System Couple (MSC) link through operation code F0.

```
03F0        0404
0001(SAORAA01,linkname)
```

This operation code will create (or modify) a table entry containing the eight-character secondary transaction definition name (SAORAA01) and the eight-character MSC link name. Whenever a secondary transaction is sent the table is checked for a match on definition name. If a match is found the CHNG call is made to the MSC link name instead of the transaction name. This routing will take place for secondary

transactions initiated from the Auditor (immediate or delayed) or secondary transactions initiated automatically through input transaction rule specifications (STX=). If successive F0 operations are performed with the same definition name, the final one will overlay the first. Table entries created during a Preaudit pass are not retained in a following Standard audit pass.

MESSAGE GENERATION OPERATIONS

Four types of messages can be generated and sent.

- Error messages
- Warning messages
- Information messages to a Display/Printer
- Information messages to a Project Group and/or USERID

Error messages are created to inform the user of an invalid condition found as the result of a field audit. This can occur as a result of actions either within the Automatic Field Assignment phase or the Field Audit/Logic phase of the Auditor. Creation of an error message is initiated through a blank specification in the Next True or Next False fields in the Operation Descriptor. For example, if the condition returned from an audit/logic routine is true and the Next True Sequence field is blank, the Auditor will mark the field in error and the four digit message number will be placed in the appropriate field entry in the Segment Layout Rule. No additional audit/logic routines will be processed against a field after an error condition has been encountered. The message number will later be used by the Transaction Driver to produce the error message. Batch and nonconversational BMP transactions print all errors on a hard-copy listing. Conversational and nonconversational MPP transactions use the message line of the Segment Display Screen, if only one message is present and will fit on the line. For conversational screens the message text must be 65 or less. For nonconversational screens it must be 50 or less. If the message text is greater than the limit or there is more than one message, the Error screen is used. The message is contained in the Message Data Base with a key of ssssnnnn.

```
ssss = Application System ID
nnnn = Message number
```

Error Message Example: If the audited field (AAAA) is less than 50 and greater than the related field (BBBB), then error message 1234 should be displayed.

```
0105          0200          Operation Descriptor 1  (check field < 50)
0001(50)                    Data Descriptor         (value 50)
0202SSXXBBBB  001234        Operation Descriptor 2  (check field >
                                                    related field)
```

Error messages can also be explicitly set on related fields by using operation code F1. Setting an error by this method is different from the way shown above in two areas.

First the audit is not terminated after the field is marked in error. This allows multiple error messages to be generated under a single audit.

```
10F1SACDFLD111117777
11F1SACDFLD212128888
12F1SACDFLD300009999
```

In the example above three fields would be marked in error before the audit terminates. One of the fields specified could also be the audited field. This operation marks the specified field in error and associates the error message number with it. A subsequent F1 operation against the same field causes the error message number to be overlayed. If a field is marked in error with this operation and the field subsequently comes up for auditing the auditing will not occur since the error flag is set.

Second this audit operation can be used on audits residing in the Message Sending leg of the Audit Data Base. Though this leg is normally used to trigger Informational messages or perform final calculations, there may be situations where errors must be flagged.

Warning messages are controlled by operation code 29. Warning messages differ from error messages in that auditing is not necessarily terminated when a warning message is flagged. Also, warning messages may be displayed in conjunction with data base updates. For example, if field AAAA is audited for three values, and the value is >1, then warning message 0001 should be displayed; if the value is >2, then warning message 0002 should be displayed; if the value is >3, then an error message should be displayed.

```
0104        0200            If field >1
  0001(1)
0229        03000001        Set warning msg and continue
0304        0400            If field >2
  0001(2)
0429        05000002        Set warning msg and continue
0504          000003        If field >3
  0001(3)                   Set error msg and terminate.
```

In this example, the warning message 0001 is subsequently overlayed with message #0002 and finally error message #0003. Multiple message numbers cannot be associated with one field since each field definition has only one area to hold the number. Operation Code 29 always takes the next true branch, so it should contain either a 00 (terminate the audit) or another sequence number (continue with additional processing).

Warning Messages can also be set on a related field by including the field name as shown.

```
0329SACDFDL104001234
```

Using multiple related fields will allow more than one warning message to be set in an audit.

Warning messages are displayed with error messages on the ERROR screen. Single warning messages are not displayed on the Segment Display Screen. If Conversational Processing is used and only warning messages are present from a Standard audit, the following message is also displayed on the screen:

```
*** ALL MESSAGES ON THE SCREEN ARE CLASSIFIED AS WARNING.  ENTER
A 'U' TO PROCEED WITH UPDATE.
```

The terminal operator may enter a 'U' which will allow the transaction to complete normally or a non-'U' which will cause the segment display screen to be redisplayed. If any error messages are present, the previous message will not be displayed. If the Auditor sets only warning messages, Message Sending phase audits are not processed. If a 'U' is entered, the Auditor will be recalled and the Message Sending phase is processed. Nonconversational or Batch processing will display/print warning messages and automatically complete the transaction if no error messages are found. If warning messages are set during Preaudit they are treated as errors and the preceding sequence does not occur.

If any data is entered in data base fields (not pseudo segments) along with the 'E' (to display the Warning message(s)), all fields marked as warning will be treated as errors. This ensures the integrity of data written to the data base when a 'U' is entered on a screen containing only Warning messages.

An informational message to the Segment Display Screen or batch register is created by operation Code D1. This message, up to 50 characters in length, will replace any system message normally displayed. For example, the normal system message at the successful completion of an add operation is '***Data Added Successfully***'. This message could be overridden as follows:

```
10D1        00
 0001(THE PARTS INVENTORY REC
```

0002 ORD HAS BEEN ADDED)

This message will be overridden if an error or warning message is subsequently triggered. In this case, the message ***ENTER 'E' TO DISPLAY ERROR MESSAGE ***' is displayed. Each D1 operation overrides any previous D1 operations.

Information messages to a Project Group and/or USERID are created in the Message Sending phase of the Auditor. This phase occurs only if no errors have occurred in the Automatic Field Assignment or Field Audit/Logic as described above. Information messages can be used to alert individuals to modifications made to a field or to the fact that critical field values are different than expected. Information messages are sent either to Project/Group mail boxes (Sign-on Profile Data Base) or USERID mail boxes (Message Data Base). These messages later can be displayed through option selection B or I on the Primary Option Menu.

A field is a candidate for Message Sending if it is a key field or flagged as changed and marked for Message Sending (MSG=Y - Rules Generator Specification). An error condition will occur if Descriptor Segments have not been created in the Message Sending phase for the candidate field.

The logic in the descriptor segments will determine whether or not a message will be sent. Message generation logic uses a blank specification in Next True or Next False field to trigger the message. If a message is required, the Auditor marks the field for message sending, and the four-digit message code (0Cnn) is placed in the Segment Layout Rule field entry. This code, which contains a one-character Format Code (C) and a two-character Message Group Code (nn), is used later to retrieve the correct Automatic Message Routing segment from the Message Data Base. The key of the Automatic Routing segment can be in one of four formats:

```
spgmxxnn - Detail by SYSID, Project/Group, Mode, Segment ID,
           Routing Code (Format Code = 0)
ss###### - Global by SYSID (Format Code = 1)
ss##pg## - Global by SYSID and Project/Group (Format Code = 2)
ss####nn - Global by SYSID and Routing Code (Format Code = 3)
```

By using this technique, routing segments can apply to an entire application system, or down to a specific transaction. The Automatic Message Routing segment and its dependents describe the message(s) to be sent and destination(s) to which they go. Refer to the "Message Data Base" section in this chapter for additional information.

Informational Message Example: If the audited field (AAAA) is updated by the USERID (111111 or 222222), then send a message (number 1234) to the USERID (333333). (SYSID=BANK)

```
0167        0200          Operation Descriptor 1  (check for update mode)
0001(5)                   Data Descriptor         (mode 5 - update)
0268          000391      Operation Descriptor 2  (check for USERID)
0001(111111,222222)       Data Descriptor         (USERID values)
```

Operation Descriptor 1 verifies that the transaction is in the update mode. If so, Operation Descriptor 2 verifies that either USERID 111111 or 222222 is currently signed on for the transaction. If so, the field is marked for message sending and the four-digit message code 0391 is saved with the field. Later, this code is used to find the appropriate routing segment in the Message Data Base. In this example, the key of the routing segment would be BA####91. This segment will specify the number of the message to send (1234) and the destination (333333).

TABLE OPERATIONS

The Auditor can process audited or related fields against table data. Functions such as decoding an argument into a value, encoding a value into an argument, or table lookup of argument or function entries, can be accomplished with Operation Descriptor codes B0-B7. All table operations are performed against entries in the Table leg of the Audit Data Base (Figure 5-1).

The Table leg is composed of two segment types. The Table Name segment (TN) contains two fields, 28 bytes long:

TABLE ID	TABLE DESCRIPTION

TABLE ID = A 6-character key which identifies the table

TABLE DESCRIPTION = A 22-character description of the table

Under each Table Name segment are the specific table entries. Each entry is contained in a Table Entry segment (TA) composed of two fields, 78 bytes long.

ARGUMENT	FUNCTION

ARGUMENT = An eight-character table argument and primary key field
FUNCTION = A 70-character value and search field

The operations associated with Descriptor codes B0-B7 access a specific table, looking for an argument or function matching an audited or related field. These operations fall into three major categories:

DECODE (B0-B1):

The decode operation translates a one- to eight-character field (argument) into a one- to 70-character field (function). Following is an example of decoding a two-character state code into a 30-character state name.

```
Audited field   SACDSTCD        (two-character NUM)
Related field   SACDSTNM        (30-character ALPHA)
FA  01B0SACDSTNM02  2222         Operation Descriptor
DF  0001(SAMPAUDTTABLES  TABLE1) Data Descriptor
```

The Operation Descriptor specifies that the value contained in the audited field (SACDSTCD) is to be used to generate an argument key of a Table Entry segment. This TA segment is in a table defined in the Data Description segment. 'SAMPAUDTTABLES ' is the 16-byte root key of the data base audit record containing the table. The table can be located within the audit record for the audited field, or, as in this example, under a different audit record. TABLE1 is the 6-byte key of the appropriate Table Name (TN) in the audit record. The final part of the generated key becomes the argument of the appropriate table entry. Since the argument key field is eight characters and the audited field is two characters, the generated key appears as 'XX ' (two-character state code with a blank filler). Once the appropriate table entry is retrieved, the Operation Descriptor further specifies that the function field should be moved to the related field SACDSTNM. The length of the move is the length of the receiving field. In this example, the first 30 characters of the TA function field will be moved to the related field SACBSTNM. If the entry is not found in the table, a false condition is returned and in this example, an error is flagged, and message 2222 displayed.

The table name can be contained in a related field instead of explicitly included in the Data Descriptor. If in the above example, the 22-character table name was contained in a field called SACDTBLE the Data Descriptor would be:

```
DF 0001(SACDTBLE)
```

ENCODE (B2-B3):

The encode operation searches the table entries until a function field is found matching the specified audited or related field. If a match is found, the corresponding argument is moved to the field specified in the Operation Descriptor. The length of the audited and related fields control both the function compare and the argument move. If no match, a false condition is returned as previously explained. The function search field (70 bytes) is built the same as the argument key field. The field contents are moved in, and blanks are filled to the right.

TABLE LOOKUP (B4-B7):

Tables can also be searched for a specific argument or function field. Operation codes B4 and B5 verify if an audited or related field is a table argument, and codes B6 and B7 a table function. Return from these routines contains a true or false condition, depending on a successful hit.

Refer to "Updating the Audit Data Base (DL/I Version)" on page 4-43 in Chapter 4, "Dynamic Rules Data Bases" and "Table Identification and Specification" in Chapter 3, "Audit Language" for information pertaining to creation and maintenance of Table segments.

DL/I DATA BASE OPERATIONS

DL/I Data base operations may be performed through the Auditor, against segments within or outside the target segment path(s). Operation Descriptor code 36 specifies that a DL/I call be performed against a segment. Operation Descriptor code 37 permits checking the DL/I status code from the previous operation, and branching accordingly. The following describes the descriptor segments:

OPERATION CODE 36

```
0136XXXXXXXXNTNF####          Operation Descriptor
0001(ID,FUNC[,I])             Data Descriptor
```

XXXXXXXX = Name of the related field containing the concatenated segment key. A keyword 'KEYFIELD' specifies that the concatenated key of the previous retrieval of this segment type should be used.

NT = Next true sequence. If the data base operation is successful (blank DL/I Status Code), a true condition is returned.

NF = Next false sequence. If the data base operation is unsuccessful (non-blank DL/I Status Code), or a delete operation is issued against a segment without eligibility, a false condition is returned. Delete eligibility is established through the Rules Generator.

= Error message number if desired.

ID = Two-character segment ID.

FUNC = Four-character DL/I function code. These function codes duplicate those used in Special Processing, and are explained in Chapter 6, "Special Processing and User Exits" under "SEGHNDLR".

[I] = constant 'I' (optional). If 'I' is specified, data base updates (INSERT,REPLACE or DELETE) will also occur immediately. If 'I' is not specified the segment is flagged for update by the transaction driver at the end of the transaction. All retrieve operations occur immediately.

Note: All updates issued during PREAUDIT or KEYAUDIT must use 'I' operand. Care should be taken when using an immediate update. If a data base is updated immediately during PREAUDIT or KEYAUDIT and data base

errors occur during the final update operations at transaction termination, the immediate updates will not be backed out since they occurred before the last sync point.

Note: If immediate updates occur in P0 or P1 phases of a Process audit and a subsequent audit error is detected, the updates may be performed again when the Auditor is reentered after the error is corrected. For this reason, all immediate updates should be issued in the P2 phase or the transaction logic should be designed such that audit errors will not occur after immediate updates are performed.

Note: The transaction driver will issue a Rollcall if data base errors are detected during final transaction updates and a data base has previously been updated. A Rollcall will not be issued if immediate updates are performed through the Auditor and subsequent audit errors or transaction/mode switches occur. It is the developer's responsibility to design the logic to handle these situations.

OPERATION CODE 37

```
0137        NTNF####            Operation descriptor
0001(GE,AM,.....XX)             Data descriptor
```

Code 37 checks the DL/I status code from the previous code 36 operation against the list in the Data Descriptor, with a true condition returned if a match is found. Non-immediate updates will always return a blank DL/I Status Code except for delete ineligibility or deleting a segment which has <u>not</u> been previously retrieved.

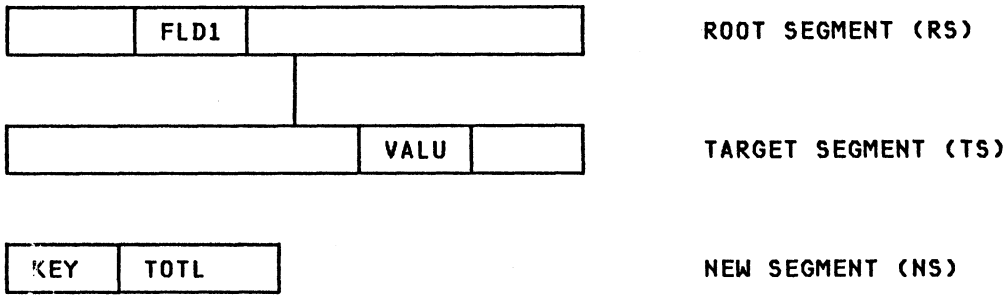
For segments to be retrieved or updated through this method, they must be described in the Input Transaction Rule (ITR) currently in control. Segments to be used in a transaction are generated in the Input Transaction Rule through the following Rules Generator parameters.

DBPATH specifies the segment(s) in the target segment path(s) to be loaded and maintained by the Transaction Driver. TSEGS specifies the segment(s) outside the target segment path(s) to be retrieved and maintained by the Auditor, or a Special Processing program.

All segments retrieved through the Auditor are read into the SPA segment area, and may be accessed, updated, or displayed as if the Transaction Driver had performed the load. Update functions are performed immediately if 'I' is specified or flagged for later update if 'I' is not specified. With Delete, the segment is first checked for eligibility, and if eligibility is not present, an 'AM' status code is set. A delete (DLET) operation performed against a segment which has not been retrieved also returns an "AM" status code. To delete or replace an unretrieved segment, the FUNC 'HDEL' or 'HREP' parameter must be specified.

Following is an example of auditor logic:

- Read all twin occurrences of a target segment (TS) and accumulate field (VALU).
- Upon completion, insert a new segment (NS) containing the accumulated total to another data base.
- The key of the new segment is derived from a field in the root segment (FLD1).



The transaction driver reads the DBPATH (RS-TS) and gives control to the Auditor. Auditing is invoked on field FLD1 by specifying PAUDIT=Y for that field. The New Segment is described in the ITR through a (TSEG=NS) parameter. The following descriptor logic is invoked:

```
0151SSTSVALU02                      Add related field to accumulator
0236KEYFIELD0103                    Read next occurrence of TS
                                    segment.  If
0001(TS,GN  )                       OK branch back to 01 otherwise
0337        04      7777            verify that all entries have been
                                    read.
0001(GA,GB,GE,GK)                   If an error quit with msg# 7777.
0463SSNSTOTL05                      Move total to new segment.
0511SSNSKEY 06                      Set up key of new segment.
0636SSRSFLD100--8888                Insert the new segment.  If OK
                                    quit,
0001(NS,ISRT)                       otherwise mark in error.
```

DB2 DATA BASE OPERATIONS

This section describes DB2 function supported through the Auditor. It includes implementation of DB2 CALLS, SQLCODE checks, SQLWARN checks, SPASQLKS sets, SPAWHERE sets, NULL sets, NULL checks, truncation checks, VARCHAR implementation, FLOAT implementation, VARLIST implementation, and NULL implementation. Refer to the IMS Application Development Facility II Version 2 Release 2 DATABASE 2 Application Specification Guide for more details concerning DB2 support.

1. Operation G3.

   Set SPASQLKS in the SPA. SPASQLKS is a one-character field. The following example sets SPASQL in the SPA to 9, which will indicate that IMSADF II key selection should use the user-defined SQL function named KSELECT9.

```
01G3        02020000        SPASQL = 9
0001(9)                      or
                            SPASQL = KSELECT9
```

   In the previous example, the High Level Audit Language will insure that the data descriptor is within the range from 3 through 9.

   The following example sets SPASQLKS in the SPA to the contents of a related field which would contain a value from 3 to 9:

```
01G3BAPSFLDN02020000        SPASQL = BAPSFLDN
                      or
01G3BAPSFLDN02020000        SPASQL = FLDN
```

   The related field must be a one-character field with type of ALPHA, A/N, or NUMERIC. The Auditor will also check to see if its value is within the range from 3 through 9. If it is not, the Auditor will generate a terminating error message. No check will be made to determine if the related field is NULL.

2. Operation G4.

   Set SPAWHERE in the SPA to point to the field which contains the host variables for the WHERE clause, in the order specified for the WHERE clause. When KEYFIELD is specified, SPAWHERE will point to the current key. No check will be made to determine if the related field is NULL. The following examples set SPAWHERE in the SPA to the offset for BAPSSTR1, which is the field holding the key selection WHERE host variables:

```
01G4BAPSSTR102020000          SPAWHERE = BAPSSTR1
                        or
01G4        02020000          SPAWHERE = STR1
```

   The following example will cause SPAWHERE to point to the current key:

```
01G4KEYFIELD0202              SPAWHERE = KEYFIELD
```

3. Operation G5.

   Test a column for NULL. A check is made on the indicator variable. The following example returns TRUE if BAPSSTR1 is NULL:

```
01G5BAPSSTR102030000          IF BAPSSTR1 NULL = ON
                        or
01G5        02030000          IF STR1 NULL = ON
                        or
01G5        02030000          IF NULL ¬= OFF
```

4. Operation G6.

   Test a column for truncation. A check is made on the indicator variable. The following example returns TRUE if BAPSSTR1 is truncated:

```
01G6BAPSSTR102030000          IF BAPSSTR1 TRUNC EQ ON
                        or
01G6        02030000          IF STR1 TRUNC NE OFF
                        or
01G6        02030000          IF TRUNC NE OFF
```

5. Operation G7.

   Set a column to NULL. A setting of the indicator variable is done. The following example sets BAPSSTR1 to NULL, i.e., it sets the indicator variable for BAPSSTR1 to indicate a NULL value:

```
01G7BAPSSTR102020000          BAPSSTR1 NULL = ON
                        or
01G7        02020000          STR1 NULL = ON
```

6. Operation G8.

   Execute a SQL call. The valid SQL functions are: SELECT, CSELECTC, CSELECTO, UPDATE, CUPDATEO, CUPDATEU, CUPDATEC, DELETE, CDELETEO, CDELETED, CDELETEC, INSERT, or a user-supplied function. No check will be made to determine if the related field holding the key data is NULL.

   The following examples execute function DEPTSELC on table ES using the key in STR1:

```
01G8BAPSSTRI02030000          IF SQL DEPTSELC BAPSSTR1 ES NOT OK
0001(ES,DEPTSELC)
                        or
01G8BAPSSTRL02030000          IF SQL DEPTSELC STR1 ES NOT OK
0001(ES,DEPTSELC)
```

   The following example executes function DEPTSELC on table ES using the current key:

```
01G8KEYFIELD02030000          IF SQL DEPTSELC KEYFIELD ES NOT OK
0001(ES,DEPTSELC)
```

7. Operation G9.

   Check SPASQLCD or SPAWARNn (where n = 0 to 7) for EQUAL value. SPASQLCD checks against a list of values in the data descriptor. SPAWARNn checks for 'W' or blank, so no data descriptor is needed. SPASQLCD is a BINARY(31) field in the SPA. SPAWARN is a one-character field subdefined as SPAWARN0 through SPAWARN7 which are each one-bit fields. The contents of SPASQLCD is a valid binary number. The contents of each SPAWARNn is either OFF which indicates blank, or ON which indicates 'W'.

   The following example checks SQLCODE for a value of -551 or -557:

```
01G9SQLCODE 02030000          IF SQLCODE = '-551,-557'
0001(-551,-557)
```

   The following example checks SPAWARN4 for a value other than 'W' (or equal to blank), which generates a NOT EQUAL for the Auditor:

```
01G9SQLWARN403020000          IF SQLWARN4 ¬= 'W'
```

8. Operation H0.

   Check SPASQLCD for GREATER THAN value. Checks SPASQLCD against a value in the data descriptor. The following example checks SQLCODE for a value greater than -551:

```
01H0SQLCODE 02030000          IF SQLCODE > -551
0001(-551)
```

   The following example checks SPASQLCD for a value less than or equal to -5, which generates a NOT GREATER THAN for the Auditor:

```
01H0SQLCODE 03020000          IF SQLCODE <= -5
0001(-5)
```

9. Operation H1.

   Check SPASQLCD for LESS THAN value. Checks SPASQLCD against a value in the data descriptor. The following example checks SPASQLCD for a value less than -551:

```
01H1SQLCODE 02030000          IF SQLCODE < -551
0001(-551)
```

   The following example checks SPASQLCD for a value greater than or equal to 5 which generates a NOT LESS THAN for the Auditor:

```
01H1SQLCODE 03020000          IF SQLCODE >= 5
0001(5)
```

VARLIST

For a SQLHNDLR operation in error, the Auditor maintains additional error table information. This includes SQLCODE (4 bytes) and SQLWARN (1 byte). This information can be used during message generation by specifying VARLIST6 for SQLCODE and VARLIST7 for SPAWARN (8 contiguous bytes).

FLOAT

The Auditor supports the FLOAT data type for moves from an eight-byte FLOAT field to another eight-byte FLOAT field, and for moves between FLOAT and A/N data to support the interface to the screens. Any other manipulation of FLOAT data is not supported in the Auditor and results in the Auditor generating a terminating error message.

VARCHAR

The Auditor supports the VARCHAR data type similar to the support of A/N data, with the addition of maintaining the halfword length.

NULL

This section describes how the Auditor checks for NULL data and what action is taken in various cases:

For REQ=YES processing, always fail the test if the field is NULL, because, by definition, NULL is an initialized value.

When calling an audit exit, the base and related field parms will be passed as usual even if they are NULL, and the exit will have the responsibility of checking for NULL.

The Auditor will support the new error table data for SQLHNDLR calls.

For Encode and Decode, no check will be made to determine if the related field holding the key data is NULL. The Auditor allows decoding into a NULL field and turns the NULL flag off.

For data move operations, if the 'from' field is NULL, the 'to' field is set to NULL if it can be NULL. If it cannot be NULL, an error is flagged. If the 'to field' is NULL, the move is done and the 'to' field is set to not NULL.

For compare operations, if either field is NULL, the compare is always unequal since a NULL field doesn't compare with anything--even another NULL field.

For the D2 operation which moves a field to SPAKEYID, no check will be made to determine if the source field is NULL.

For op code D3 which moves SPAKEYID to a field, the Auditor allows the move and turns the NULL flag off.

For twin loops, when attempting to set the from/to numbers from fields, no check will be made to determine if the field is NULL.

When attempting to set the PCB number from a NULL field, the Auditor will do the move without checking for NULL, because this would not be the normal situation. If the user does need to check in this case, he can use op code G6.

When assigning the current date to a field, if the field is NULL, it is set to not NULL after the assignment.

When assigning the current time to a field, if the field is NULL, it is set to not NULL after the assignment.

For op code 17 which checks a bit, if the field is NULL, the test will fail.

For op codes 18 and 19 which set bits, if the field is NULL, the field will be set to not NULL.

When testing for A, A/N ,N, etc..., if the field is NULL, the test fails automatically.

When doing range checking, the test always fails when the field is NULL, since the range cannot contain NULL.

When checking a list, the test always fails when the field is NULL, since the list cannot contain NULL.

For DL/I operations, no check will be made to determine if the related field holding the key is NULL.

For op code 41--Compare for characters not in list always fails for a NULL field since NULL means unknown.

For op code 43--Character mask operation, always fails for a NULL field since there are no characters in the field available for comparison.

For Arithmetic: If a source field is NULL, the target field is set to NULL without doing any arithmetic. If the target field is NULL, the operation is done and the field is set to not NULL.

When assigning system fields to fields, if the field being assigned is NULL, the assignment is done and the field is set to not NULL.

When checking for DBCS or MIXED containing DBCS, if the field is type DBCS, no NULL check is made, and the test passes. If the field is MIXED, a NULL test is made. If the MIXED field is NULL, the test fails since it does not contain DBCS data.

For concatenation, if the target field is NULL, do the operation and set the target field to not NULL. If a source field is NULL, the concatenation is not done, and the target field is set to NULL.

For substringing, this will work just like any assignment. For G1 and G2 op codes, if the source field is NULL, the target field is set to NULL. For G1, if the target field is NULL, the move is done and the target field is set to not NULL. For G2, if the target field is NULL, it is a NOP. Note that when the position and length are in related fields, these related fields cannot be NULL. There will be no check made to determine if these fields are NULL.

Op code G3--Set SPASQLKS. No check will be made to determine if the related field is NULL.

Op code G4--Set SPAWHERE. No check will be made to determine if the related field is NULL.

Op code G8--DB2 operations. No check will be made to determine if the related field holding the key is NULL.

For the ALPHA, NUM, or DATE check, the test always passes if the field is NULL.

The following error messages are added:

```
0134  INVALID DATA TYPE FOR SUBSTRINGS
0138  INVALID USE OF NULL DATA
0140  INVALID MOVE TO SPASQL
```

TWIN OCCURRENCE PROCESSING

This section deals with Audit operations which are available to assist the user in processing multiple occurrences of the same segment type (twin processing). Since IMSADF II treats each segment ID as a unique segment definition, the normal technique used to define multiple segment occurrences is to redefine the segment layout with as many unique segment IDs as required. This technique creates a Segment Layout Rule for each redefinition with identical field definitions in each. These definitions are included in a transaction by specifying them as TSEGS.

The processing of these multiple definitions through the Auditor usually consists of logic to retrieve/update multiple occurrences and to perform edit or processing logic against fields within the segments. Normally this type of processing is repetitive in that the same field(s) in each segment definition are used. The logic is the same for each segment becoming unique only by the field name referenced. For example three twin segments are defined as segment IDs (XX,YY,ZZ). Each consists of one field called FLD1. With a System ID of SAMP, the logic to check FLD1 in each segment for a 1 would be:

```
0132SAXXFLD102  9999    Check FLD1 in XX segment
0001(1)
0232SAYYFLD103  9999    Check FLD1 in YY segment
0001(1)
0332SAZZFLD100  9999    Check FLD1 in ZZ segment
0001(1)
```

As you notice, the operations are identical except that the segment ID changes in each related field reference. Operations are available which allow the type of processing shown above to be performed in one set of operations instead of three. These twin processing operations allow the user to define the twin segments to be processed in a 'twin loop'. A twin loop is defined as a group of audit operations in which all references to an audited field or related fields residing in one of the

twin segments is automatically adjusted to the current iteration of the loop. In the above example a twin loop defined for segments XX, YY, and ZZ would need only one operation to perform the check of three fields.

```
0432SAXXFLD105  9999
0001(1)
```

On the first iteration FLD1 in the XX segment will be checked. If not '1' the audit will be terminated with an error. If it is 1 the next iteration checks FLD1 in the YY segment. On the last iteration FLD1 in the ZZ segment is checked. In this example the audited field is not in the twin occurrences so if any of the three fields is not 1 the audited is terminated with an error 9999. If FLD1 is the audited field the following operation in a twin loop could be used.

```
0422        05  9999
0001(1)
```

Since FLD1 is an audited field it is assumed that FLD1 in each segment is an audited field. Using the above operation in a twin loop would cause an error to be flagged on each FLD1 field which did not equal 1.

Before showing additional examples, the operations to set and control twin loop processing will be explained in detail. These four operations can be used in conjunction with all other audit operations to construct routines to process twin occurrences. These operations are valid for all audit calls (Key, Preaudit or Standard) and all audit phases.

The SETTWIN operation is used to define the twin segment IDs to be used in the following twin loop processing. The format of the operation is:

```
01E4        0202
0001(XX,YY,ZZ)
```

The accompanying Data Descriptor lists all segment IDs that are to be considered as twins, in the order in which they are to be processed. Up to 100 IDs may be specified in a single operation. A subsequent E4 operation will replace any current list of IDs. All IDs specified must be defined to the transaction as either DBPATH or TSEG segments. They may be either data base segments or pseudo segments. The list created by the SETTWIN operation is used when processing audit operations bounded by a DOTWIN and ENDTWIN operation.

The DOTWIN operation is used to the define beginning of a twin loop. In addition the operation is used to define the starting and ending segment IDs (defined by the SETTWIN operation) to process.

```
02E5        0305
0001(x,y)
     or
0001(related1,related2)
```

The accompanying Data Descriptor contains two parameters, either explicitly defined (x,y) or dynamically set in two related fields (related1,related2). The parameters are numbers from 1 to the number of IDs in the SETTWIN operation. The first parameter specifies the first ID to process and the second parameter specifies the last ID to process. If related fields are used to hold the parameters, they must have a type of DEC, NUM, PD, or BIN. The Next True sequence points to the next-operation and the Next False sequence points to the ENDTWIN operation.

The ENDTWIN operation specifies the end of a twin loop causing control to be passed back to the DOTWIN operation

```
05E6        0606
```

The Next True and Next False sequences point to the next operation. The SETARRAY operation defines the first in a series of fields, in a pseudo segment, which are associated one for one with each twin segment.

```
01F5SACDFLD10202
```

The related field names the first field of the array.

This field is associated with the first twin segment during iteration one. The next field is associated with the second twin segment during iteration two and so on.

In order to better explain the use of these operations we will use the following example:

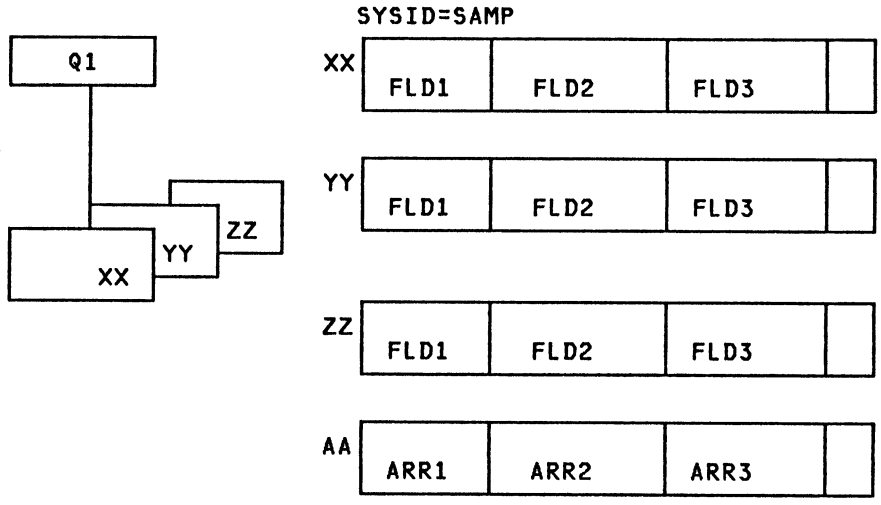


Figure 5-5. Twin Processing Example

Figure 5-5 shows three twin segment definitions which are identical except for the segment ID. In addition, a pseudo segment (AA) is defined with three fields. Field ARR1 is associated with segment XX, ARR2 with YY, and ARR3 with ZZ. Under normal Audit coding the logic to perform an audit test on the first field in each segment would require three routines with SAXXFLD1,SAYYFLD1 and SAZZFLD1 as three audited fields. By using the operations shown above one routine can be developed to perform the required logic.

The following example shows the coding required to audit FLD1 in each segment and set an error if FLD1 is greater than FLD2, less than FLD3 and equal to its corresponding ARR(x) field. The field names shown in this example have no specific meaning other than for illustrative purposes.

```
01E4          0202        1 SETTWIN
0001(XX,YY,ZZ)
02F5SAAAARR10303          2 SETARRAY
03E5          0407        3 DOTWIN 1 to 3
0001(1,3)
0402SAXXFLD20507          4 Test GT. FLD2
0503SAXXFLD30607          5 Test LT FLD3
0630SAAAARR1  073333      6 Test EQ ARR(x)
07E6          0808        7 ENDTWIN
08                        8 Additional Audits
09
```

After reading operation 1 the Auditor stores the list of segment IDs for later processing when a twin loop is encountered. A twin loop is defined as all operations bounded by a DOTWIN/ENDTWIN operation. It is assumed for this example SAZZFLD1 is the audited field. Only one of the FLD1 definitions need be specified as an audited field although all could be so specified. Operation 2 defines the first field of an array and associates it with the first twin definition. Operation 3 causes the Auditor to initiate a twin loop. The start and end ID numbers (1,3) are stored. In this example the (1,3) specifies that the first time through the twin loop the audited field will be adjusted to the FLD1 field in the XX segment. Also any related field references will be adjusted to the XX segment definition if appropriate. The 3 specifies

looping will continue until the third segment ID (ZZ) has been processed.

If the Data Descriptor had contained a (2,3) the twin loop would only have processed the YY and ZZ segments. A (1,2) would have eliminated the processing of the ZZ segment. Operations 4 and 5 are testing the audited field for the current iteration against the related fields for the current iteration. On the first iteration all pointers are to definitions in the XX segment, then YY and finally ZZ. If a related field, which is not in one of the twin occurrences, is referenced its pointer will not be affected by the iteration. Operation 6 tests the audited field for the current iteration against the field defined as the first field of the array. Since this field is defined in the SETARRAY operation, it is also adjusted by the iteration. Notice that the field in SETARRAY cannot be subdefined. On the first iteration, the pointers are to field ARR1, on the second iteration to field ARR2, and on the third iteration to field ARR3. Operation 7 signifies the end of a twin loop and causes the Auditor to automatically branch back to the previous DOTWIN operation. Each time the DOTWIN operation is processed the iteration counter, field pointers and segment pointers are adjusted. As long as the specified number of iterations are not complete the Next True branch is taken. When all iterations have been performed the Next False branch is taken. The Next False sequence number points to the ENDTWIN operation. Since all iterations are complete, the next true branch of the ENDTWIN operation is taken. These branches are created automatically through the High Level Audit Language. Since FLD1 was both an audited field and contained within the twin definitions, each iteration causes the appropriate definition of FLD1 to be considered the audited field. If during the twin loop processing an error is detected the current audited field is marked in error and control is passed back to the DOTWIN operation. The next iteration is then processed with the next audited field. At the end of the last iteration, the audited field pointer is reset to the original audited field and processing continues. In the above example any or none of the FLD1 definitions could have been marked in error with a 3333 message. If the audited field is in the twin occurrences, auditing continues after the twin loop even though one or all of the audited fields is marked in error.

If the audited field is not in the twin definitions the first error will cause the audit to be terminated.

```
01E4        0202          1 SETTWIN
0001(XX,YY,ZZ)
02E5        0305          2 DOTWIN 1 TO 3
0001(1,3)
03C2SAXXFLD20405          3 TEST GT
0001(SAXXFLD1,SAXXFLD2)
04C3SAXXFLD3  053333      4 TEST LT
0001(SAXXFLD1,SAXXFLD3)
05E6        0606          5 ENDTWIN
```

The above example shows a twin loop routine where all the fields are related fields. In this example the first time FLD1 is > FLD2 and < FLD3 the audit is terminated with the audited field marked with error 3333.

DL/I calls can be included in a twin loop to allow twin retrieval, processing and update in one routine.

```
01E4        0202          1 SETTWIN
0001(XX,YY,ZZ)
02E5        0306          2 DOTWIN 1 TO 3
0001(1,3)
0336SACDTWIN0407          3 Perform a GN for a twin
0001(XX,GN  )
04C4SAXXFLD20505          4 Bump FLD2 by 1
0001(SAXXFLD2,1)
05C3SACDTWIN06  3333      5 Replace twin
0001(XX,REPL,I)
06E6        0707          6 ENDTWIN
```

On each iteration this example will retrieve a twin occurrence with a GN. Assume SACDTWIN contains the concatenated key of the last segment retrieved. If no more occurrences are available a branch is taken out

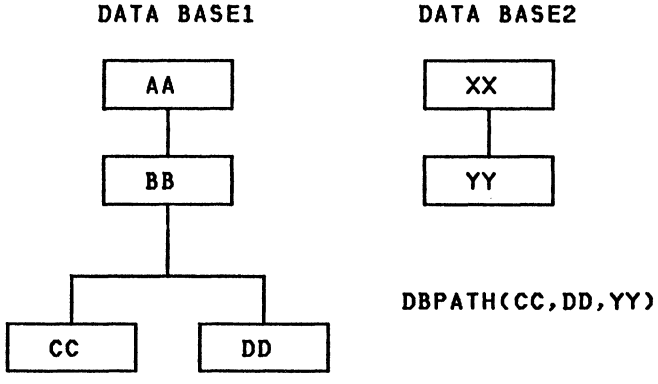
of the loop. Otherwise FLD2 in the segment is increased and the segment is updated. Each of these twin occurrences could then be displayed on a screen. This example will retrieve and update up to three twin occurrences each time it is invoked.

If twin loop routines call subroutines or audit exits the routines called are assumed to be within the loop and as such take on the pointers of the current iteration.

If a branch is taken from inside to outside the twin loop (before the DOTWIN or after the ENDTWIN) the iteration process is terminated and normal audit processing continues. If a branch is taken into a twin loop (DOTWIN operation is not processed) the iteration process is not initiated. Processing continues sequentially from the point of entry. The next operation after the ENDTWIN operation is executed instead of a branch back to the DOTWIN.

TRANSACTION SWITCH OPERATIONS

The Auditor may be used to switch from one IMSADF II transaction to another. This function is used in the same manner as a return code 5 from a Special Processing Routine (Chapter 6, "Special Processing and User Exits"). There are four Operation Codes (D2-D5) which may be used in a transaction switch. Code D4 specifies that the two-character transaction ID (contained in a Data Descriptor) replace the current one. Code D5 specifies that the one-character mode (contained in a Data Descriptor) replaces the current one. Each of these operations will trigger a subsequent transaction switch when control is returned to the transaction driver. If an audit error or warning message is found, the transaction switch will not occur and the transaction ID and mode will be reset to their original values. In some cases, the DBPATH concatenated key(s) may also need altering. If so, code D3 will move data from SPAKEYID into a related field for modification and code D2 will move data from a related field to SPAKEYID. The length of the move is controlled by the length of the related field. Data is moved to and from SPAKEYID in a character format. SPAKEYID is constructed as follows:



SPAKEYID = KEY(AA) KEY(BB) KEY(CC) KEY(DD) KEY(XX) KEY(YY)

The concatenated keys are formatted in the order of definition in the DBPATH parameter. If parent keys are previously inserted (KEY(AA), KEY(BB)) they are not redefined as shown with KEY(DD). Each segment key is contained only once in SPAKEYID. There are NO blanks or separators between keys. If an audit or warning message is found, SPAKEYID will be reset to its original value. A transaction switch initiated by the Auditor causes the following events to occur after returning control to the transaction driver.

- Final update of segments in the current transaction.
- Process the new transaction in the following order:

1. Primary and secondary key Selection and audits.
2. Preaudits
3. Display of the Segment Display Screen.

The message displayed on the screen is either a standard message, 'ENTER DATA FOR UPDATE', or a tailored message created through option code D1 at the time the Auditor initiated the transaction switch.

Following is an example of Transaction Switch operations:

```
01D4        02      Change Transaction
 010001(XX)          ID to xx
02D5        03      Change Transaction
 020001(5)           Mode to 5
03D3ssxxffff04       Get SPAKEYID
04D2ssxxffff00       Set SPAKEYID
```

MODIFYING PCB NUMBER THROUGH THE AUDITOR

Each segment generated in a transaction (DBPATH or TSEG) is associated with a relative PCBNO within the transaction PSB. In cases where large files are split across multiple data bases the specific PCBNO for the segment(s) may not be known until the key values are entered. At this point it may be useful to dynamically change the PCBNO associated with a segment. The Audit Language Statement "xx PCBNUM = yyy" generates:

```
01E9        0202
0001(xx,yyy)
```

The E9 operation will perform as follows. The segment ID specified by xx is used to search the Input Transaction Rule for a match on a segment (ITRSEG) definition. If one is not found an error is flagged.

Once the segment is found, it is necessary to point it and related segments to the same PCB. Therefore, the current PCBNO associated with the segment is used on a secondary search of the Input Transaction rule. This search checks each path (ITRPATH) and segment (ITRSEG) entry for a match on PCBNO. Each match is changed to the new relative PCBNO specified by yyy. The relative PCBNO (yyy) should correspond to the user data base position in the PSB (1 to n where n is the number of user data bases).

In the above example segments AA and BB are associated with relative PCBNO 1 and segment CC is associated with relative PCBNO 2 through the Rules Generator. If during execution segments AA and BB must be associated with PCBNO 3 the following operation is used.

```
01E9        0202
0001(AA,003)  or   0001(BB,003)
```

After the operation segments AA and BB would be associated with PCBNO 3 while segment CC remains associated with PCBNO 2. This association will remain in effect as long as the transaction is processing or until a subsequent E9 operation modifies it. In the above example if segment CC was originally associated with PCBNO 3, all three segments would be associated with PCBNO 3 after the operation. A subsequent operation would cause all three to be changed.

The relative PCBNO (yyy) can be dynamically set in a related field and referenced. For example:

```
01E9SACDFLD10202
  0001(xx)
```

In this example, the related field, SACDFLD1, contains the relative PCBNO. The related field must be NUM, PD, BIN or DEC.

ROLLCALL OPERATION

If immediate updates are requested through the Auditor, an error condition may arise which requires a ROLLCALL operation. Operation Code C8 will cause this to occur. In addition, a spanned Data Descriptor may be included which will cause the contents to be outputted to the entering terminal. For example:

```
10C8
0001 (AN ERROR HAS OCCURRED
0002 CAUSING A BACKOUT OF ALL
0003 FILE UPDATES)
```

will cause the message to be displayed to the terminal (DISPLAYE) and a ROLLCALL to occur. No further processing occurs after the operation. If a Data Descriptor is not present only the ROLLCALL is issued.

SETTING FLAGS THROUGH THE AUDITOR

The SETFLAG operation, described under Automatic Data Base updating routines in Chapter 6, "Special Processing and User Exits" may also be invoked through operation code E2. The segment delete, retrieve, or changed flags may be set ON or OFF. Code E2 requires a single Data Descriptor in the following format:

```
0001(ID,FLAG,SETTING)
ID = two-character segment ID
FLAG = Character flag type
                D = delete
                R = retrieve
                C = change
SETTING = one-character:  0 = OFF, 1 = ON
```

Changing these segment flags will affect data base updates at the termination of the transactions as shown in Figure 5-6.

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 5-6. Segment Flag Processing

There are two special situations which can be controlled through the SETFLAG operation. The first is a mode 2 or 4 transaction where the target segment should not be inserted. The second is a mode 1 or 3

transaction where the target segment should not be deleted. By setting the flags on the target segment as shown in Figure 5-7, the update operation performed against them can be eliminated.

Changed Flag	Retrieved Flag	Delete Flag	Resultant Action
OFF	ON	OFF	Target Segment Not Inserted
OFF	OFF	OFF	Target Segment Not Deleted

Figure 5-7. Special Update Situations

TESTING FOR ANY ERROR DURING THE CURRENT AUDITOR CALL

A test can be made to see if there has been any previous error during the current call to the Auditor. It does not check for errors in previous calls to the Auditor. The format of the descriptor is:

```
01H5        02030000

       OR

01H5        03020000
```

The operation branches to the next true sequence if there has been an error, otherwise it branches to the next false sequence.

CHECKING FOR RETRIEVED SEGMENTS

Code D0 checks specific segment IDs to determine if the segments have been retrieved. A Data Descriptor contains a list of one or more two-character segment IDs.

0001(ID1,ID2,ID3.... etc.)

All data base segments in the transaction may be included. If <u>any</u> segment in the list has been retrieved, a true condition is returned.

INITIALIZING DATA AREAS

One or more segment areas can be reset to an initialized format with Code C9. This is useful if twin occurrences are read into segment areas over multiple iterations. In this situation, it is important to insure that segment areas not receiving a new twin occurrence are reset. Initialized values for specific field types are explained earlier in "Automatic Field Assignment Phase" on page 5-5. The Data Descriptor has the following format:

0001(ID1,ID2,ID3.... etc.)

BIT AND SWITCH OPERATIONS

A field, defined as type BIT, is processed with operation codes (17-19). All other compare, move or arithmetic operations are invalid for this field type. A bit within a byte can be set ON (Code 18), or OFF (Code 19) or tested (Code 20). The specific bit within a byte is determined by the BITOFF parameter specified to the Rules Generator.

Nine internal switches are available within the Auditor for decision processing. These switches are turned OFF at the beginning of each of the four Auditor calls. They may be used to control audit flow between different legs or fields. The nine switches are set ON, set OFF, and tested with Codes 26-28. The switch number is placed in the Related Field Name (free form) as follows:

```
0126SW1       02          (Either of these examples is valid
0227SWITCH1   03          if a valid number (1-9) is specified.)
03281         04
```

A Data Descriptor is not used.

KEY AUDIT OPERATIONS

Primary and Secondary Key Audit calls may use all Operation Codes except the following:

- Warning messages (Code 29)
- Secondary transaction generation (Codes 08, F0)
- Screen Attribute modification (Codes A9, F3, F4)

Operation codes D6 and E3 are applicable only to a Secondary Key Audit call. As each segment occurrence is retrieved, the Auditor is called to perform processing. If the specific segment should not be displayed, Code D6 will cause the transaction driver to delete it from the Secondary Key Selection screen. Through this technique, the display of a segment or blocks of segments may be prohibited to a user.

```
01D6        00     Prohibit display of current segment occurrence
```

If the specific segment should not be displayed and no more segments should be retrieved for Secondary Key Selection, code E3 will cause the Secondary Key Selection screen to be displayed with all segments retrieved up to that point. This code acts as if a 'GE' or 'GB' status code was returned on the segment retrieve.

```
01E3        00   Prohibit display of current segment occurrence
                 and terminate Secondary Key Selection retrieval.
```

Operation code F2 can be used to separate Key Audit operations from Preaudit or Standard Audit operations if a Format 1 root key is used. Reference Format1/Format2 keys under the AUDIT/LOGIC RULES section. This code will take the Next True branch if the Auditor call is either a Primary or Secondary Key Audit and the Next False branch if not. This operation will be automatically generated by the High Level Audit Language when KANAME = ALT is specified.

```
01F2        0204    If Key Audit go to 02 else go to 04
02 .                Key Audit
0301        0000              operations
04 .                Standard
05 .                        Audit
06 .                             operations
```

If a DBPATH segment is retrieved through a Primary Key Audit the segment retrieve flag must be set OFF. This will allow the transaction driver to retrieve the segment during normal key selection processing.

PREAUDIT OPERATIONS

Auditing can be forced on a field prior to the display of a screen (Conversational) or the mapping of transaction input (Nonconversational and Batch). Since Preaudit indicators are carried in the Input Transaction Rule, the control of this type of processing is at a transaction level. Some uses of Preaudit might be to control screen display, read additional segments outside the path, or perform computations for display.

The example below shows the use of Preaudit and Standard Audit logic on a single field (SACDCLOR). In this example, a field is checked during the Preaudit pass to determine if it should be protected. If not, on the Standard Audit pass, it is rechecked to validate the modification.

```
0116        0204        If preaudit go to 02 else go to 04
0268        0300        If not USER01 quit
0001(USER01)
```

```
03A9        0000        else protect the field
0001(SACDCLOR,PROT,1)
0421        00  1111    If modified field is within
0001(10,200)            range quit else error msg. 1111
```

Operation Descriptor code 16 is used to control which processing takes place on a pass. If the field is protected on the Preaudit pass, then no modification can be made. Subsequently, upon entry of the screen, the changed flag will not be set for the field and the Standard Audit will not occur. The Rules Generator parameters for this field would be PAUDIT=Y (ITR) and AUDIT=Y (SLR). This operation is automatically generated by the High Level Audit Language.

DBCS SUPPORT

The Auditor recognizes DBCS (double byte character set) characters. The following paragraphs describe DBCS operations in the Auditor:

Op Code F7 sets field-outlining attributes on a display field. This operation is not valid for non-DBCS terminals. This operation uses descriptor data with two possible forms. The first form specifies the target field name and a four byte outlining specification enclosed in apostrophes. The second form specifies the target field name and the name of a related field that contains the outlining specification. The operation determines the difference between the forms by the location of the terminal right parentheses.

Op code F8 returns true if a field contains DBCS data or if the field type is DBCS.

Audit operations that compare or assign data will also deal with DBCS data. In either case, the Auditor will be sensitive to the contents of a MIXED field. A mixed field with pure EBCDIC contents has no DBCS data. A mixed field with pure DBCS contents contains the framing SO and SI characters, the DBCS data between the characters, and one EBCDIC blank on the end if the field has an odd length.

Following is a more detailed description of auditor support for DBCS.

DBCS Moves and Compares

To understand the following discussion of moves and comparisons, it is necessary to define "pure" data. A MIXED field will be considered pure DBCS if it contains only DBCS data (enclosed in SO/SI), and optionally a trailing EBCDIC blank for mixed fields with an odd length. Note that a MIXED field which contains only DBCS data except for a trailing EBCDIC character other than blank will not be considered pure. The intent is to allow fields a way of being pure when DBCS data is moved into a MIXED field that has an odd length. A MIXED field will be considered pure alphanumeric if it contains only EBCDIC data (no SO/SI). The tokenizer routine (V42) will return a flag indicating when contents are pure DBCS or pure A/N.

DBCS MOVES: The following rules define the handling of moves:

1) DBCS to DBCS — Move byte for byte starting from the left. Truncate or pad with DBCS blanks if necessary. Both fields assumed to be even lengths.

2) MIXED to MIXED — Pad on the right with EBCDIC blanks if necessary. If the receiving field is shorter, the value will be truncated to preserve SO/SI pairing. Trailing EBCDIC characters will be truncated. If the truncation point is between SO/SI characters, DBCS characters will be dropped from the right end of the DBCS string until the value will fit. Then if necessary, the value will be padded with a single EBCDIC blank.

3) DBCS to MIXED — Frame with SO/SI and truncate or pad with DBCS blanks on the right if necessary. If the MIXED field has an odd length, the value will be padded on the right with a single EBCDIC blank.

4) A/N to MIXED — Truncate or pad with EBCDIC blanks on the right as necessary.

5) MIXED to DBCS — When the MIXED field contains pure DBCS data, strip off the SO/SI (and trailing EBCDIC blank if necessary) and truncate or pad with DBCS blanks on the right as necessary. When the MIXED field does not contain pure DBCS data, an error message is generated stating an illegal move was attempted. User must use an audit exit in order to move impure data.

6) MIXED to A/N — When the MIXED field contains pure EBCDIC data, truncate or pad with EBCDIC blanks on the right as necessary. When the MIXED field does not contain pure EBCDIC data, an error message is generated stating an illegal move was attempted. User must use an audit exit in order to move impure data.

The following exercises define how moves will be handled in the Auditor:

Consider the following data base fields:

DB FIELD	LENGTH	TYPE	CONTENT in HEX
A1	1	A/N	"C4"
D2	2	DBCS	"41 42"
M4	3	MIXED	"C1 C2 C3 C4"
D4	4	DBCS	"61 62 71 72"
M5	5	MIXED	"0E 41 42 0F C1"
M6	6	MIXED	"C1 0E 41 42 0F C2"
M7	7	MIXED	"40 40 40 40 40 40 40"
M7P	7	MIXED	"0E 41 42 51 52 0F 40"
M8	8	MIXED	"C1 0E 41 42 51 52 0F C2"
M9	9	MIXED	"C1 C2 C3 0E 41 42 51 52 0F"
D10	10	DBCS	"41 42 41 42 41 42 41 42 41 42"

Given the following assignment statements on the above data, the contents of the receiving field is shown as it will appear after the audit operation:

1. M4 = A1
   M4 = "C4 40 40 40"

2. A1 = M3
   A1 = "C1"

3. M5 = A1
   M5 = "C4 40 40 40 40"

4. A1 = M5
   A1 = "C4" => error--impure data

5. D4 = A1
   D4 = "61 62 71 72" => error condition--illegal move.

6. A1 = D4
   A1 = "C4" => error condition--illegal move.

7. D2 = D4
   D2 = "61 62"

8. D4 = D2
   D4 = "41 42 40 40"

9. M4 = D2
   M4 = "0E 41 42 0F"

10. M5 = D4
    M5 = "0E 61 62 0F 40"

11. D10 = M8
    D10 = unchanged = > error--impure data

12. D10 = M7P
    D10 = "41 42 51 52 40 40 40 40 40 40"

13. M9 = M8
    M9 = "C1 0E 41 42 51 52 0F C2 40"

14. M7 = M8
    M7 = "C1 0E 41 42 51 52 0F"

15. M6 = M8
    M6 = "C1 0E 41 42 0F 40"

16. D4 = M8
    D4 = unchanged ==> error--impure data

17. D4 = M7P
    D4 = "41 42 51 52"

18. M8 = D4
    M8 = "0E 61 62 71 72 40 40 0F"

DBCS COMPARES The following rules define the handling of compares:

1) Same Type — Compare on shorter length.

2) DBCS to MIXED — When the contents of the MIXED field are pure DBCS, strip off the SO/SI and trailing EBCDIC blank if necessary, then compare on shorter length. When the contents of the MIXED field are not pure DBCS, the fields do not compare.

3) A/N to MIXED — When the contents of the MIXED field are pure A/N, compare on shorter length. When the contents of the MIXED field are not pure A/N, the fields do not compare.

Consider the following data base fields:

DB FIELD	LENGTH	TYPE	CONTENTS
M6	6	MIXED	"0E 41 42 51 52 0F"
M7	7	MIXED	"C1 0E 41 42 51 52 0F"
M8	8	MIXED	"C1 0E 41 42 51 52 0F C2"
D4	4	DBCS	"41 42 51 52"
D6	6	DBCS	"41 42 51 52 61 62"
A1	1	A/N	"C1"

1. Fields M7 and M8 compare.
2. Fields M6 and M8 do not compare.
3. Fields M6 and M7 do not compare.
4. Fields D4 and D6 compare.
5. Fields M6 and D4 compare.
6. Fields M7 and D4 -- error--impure data.
7. Fields M7 and A1 -- error--impure data.
8. Fields D4 and A1 -- error--comparing unlike data types.

REQ=YES

Check for blanks if DBCS or MIXED (like A/N).

Op code C8

Message text for ROLL CALL message may be DBCS.

1. Length of a DBCS field or the DBCS portion of a mixed field must be an even number.

2. A mixed field without SO/SI is treated just like an A/N field.

3. DBCS fields cannot contain SO/SI.

4. DBCS data in data descriptors will always contain the SO/SI, i.e. they will be in MIXED field format. There may be commas separating the values. The HLAL does not have access to the SPA in order to tell what the field type is.

DBCS Restrictions

OP CODE 41: Op code 41 is not supported for DBCS. The common error message will be set in this routine indicating that an operation not supported for DBCS was attempted on a DBCS field.

OP CODE 43: Op code 43 is not supported for DBCS. The common error message will be set in this routine indicating that an operation not supported for DBCS was attempted on a DBCS field.

TABLE OPERATIONS: All table operations (B0 through B7) use only table names in the data descriptors, and table names may not contain DBCS characters.

RELATED FIELD NAMES: The operations which do moves and compares between related fields (C0 through C3) use only related field names in the data descriptors, and field names may not contain DBCS data.

OP CODES C9 AND D0: Op codes C9 and D0 perform segment initialization and retrieve checks respectively. Their data descriptors contain only segment ID's which cannot be DBCS.

SYSTEM FIELDS: Op codes D2 through D5 only contain MODE and TRXID in the data descriptors--they cannot contain DBCS data. Op codes 66 through 68, B8, B9, D7 through E1, E7 and E8 contain only SYSID, USERID, TRX ID, PROJECT GROUP, TRX MODE, or LTERM in the data descriptor, and none of these can contain DBCS data.

TWIN PROCESSING: Twin processing op codes (E4, E5, E6, and F5) contain only segment ID's in the data descriptors. They cannot be DBCS.

PCB NUMBER: Op code E9 modifies the PCB number and its data descriptor must be a numeric character--not DBCS.

SECONDARY TRANSACTIONS: Op codes for secondary transactions (08 and F0) contain only ITRMOD names in the data descriptors. They cannot be DBCS.

SUBROUTINES: Op code 20 references an audit record in the data descriptor. It cannot contain DBCS data.

DL/I CALLS: Op code 36 performs a DL/I call and its data descriptor contains only segment id, function code, and immediate indicator. Op code E2 performs a SETFLAG operation and its data descriptors contain only segment id, flag indicator, and setting. None of this data can be DBCS.

STATUS CODE: Op code 37 checks the DL/I status code, and its data descriptor must contain alphanumeric status codes--not DBCS.

ARITHMETIC OPERATIONS: The arithmetic op codes are only concerned with numbers--not DBCS data.

SET ATTRIBUTES: The set-attribute op codes will only contain field names and attributes in the data descriptors--they cannot contain DBCS data.

NATIONAL LANGUAGE SUPPORT: The new op code added to support NLS is F6 which returns a true indication if the SPAULANG value in the SPA matches one of the values found in the data descriptor. Except for this single op code, the Auditor remains unchanged in order to support NLS.

STATIC AUDIT/LOGIC RULES

As discussed previously, dynamic Audit/Logic rules are maintained as either segments in a DL/I data base or rows in DB2 tables. These records can be updated online to allow instant changes to audit/logic routines if required.

Since each Operation Descriptor and Data Descriptor is a separate record in the data base, the overhead of accessing the rules may be excessive for some applications. This problem could exist if many fields were audited and/or the audit/logic routines consisted of numerous descriptor segments.

To enhance performance, Audit/Logic rules can be created in a load module format and placed in the rule load library. The static rules will contain all the descriptors needed by a transaction. The Input Transaction Rule will specify whether a static rule (from a load library) or dynamic rule (from the Audit Data Base) should be used. The Auditor will either load the rule or access the Audit Data Base as directed.

Static Audit/Log c Rules can be created in two ways. First, they can be created directly from the Audit Language compiler. This generation is controlled by a JCL execution parameter and produces a static rule directly from the Audit Language source statements. Refer to "Using The Audit Language Procedure" for additional information on this method of creating a static audit rule.

The second method, is used when the audit operations are already load loaded in a DL/I data base. Note: If your installation uses DB2 tables to hold audit operations, the first method of creating static rules must be used. No program is available to extract dynamic audits from a DB2 table.

A batch program (distributed with the system) is used to extract specified audit routines from the Audit data base and create rule input for the Rules Generator. The user specifies which static rules should be generated through input to the batch program.

Each static rule contains all the descriptors contained within an Audit Group code. The Audit Group code is the first eight bytes of the root key for an Audit/Logic record. The Audit Group code is also associated with an Input Transaction Rule.

If static rules are being used, the appropriate flag in the Input Transaction Rule must be turned on. This is accomplished by specifying LRULE=Y or LRULE=ALT on the Rules Generator SYSTEM or GENERATE statement that describes the transaction. Upon entry, the Auditor module will determine if a static rule is needed and, if so, will use the System ID and Audit Group code to retrieve the appropriate load module. If subsequent Operation Descriptors call for branching to Operation Descriptors in another static rule, or table operations against table entries in another static rule, the static rules are dynamically loaded as needed. If audits for some or all fields in the transaction are not in the static rule and LRULE=ALT is specified, the dynamic Audit Data Base is accessed. If not found there, an error is flagged. If LRULE=ALT is specified and a static rule is not present, the dynamic Audit Data Base will be accessed.

CREATING STATIC AUDIT/LOGIC RULES

Static Audit/Logic rules can be created in either the IMS/VS batch or BMP environments through a batch application program distributed with the system. The JCL procedure is provided as a member in IMSADF.JCLLIB.JCL. This procedure (MFCB) will execute the batch program (????BDCT) in an IMS/VS batch environment.

Input to the batch transaction driver is an OS/VS sequential file with a DDNAME of TRANSIN. Each input record contains a transaction name and control information that specifies the Audit Group(s) for which static rules are to be created. The following is a layout of the card images as they would appear in the TRANSIN file.

Column	Length	Description
1	8	Transaction Name (????B5AM)
9	1	blank
10	8	Parameter 1 (eight-character Audit Group code)
18	1	blank
19	8	Parameter 2 (eight-character Audit Group code)
27	1	blank
28	8	Parameter 3 (eight-character Audit Group code)
36	1	blank
37	8	Parameter 4 (eight-character Audit Group code)
45	1	blank
46	8	Parameter 5 (eight-character Audit Group code)
54	1	blank
55	8	Parameter 6 (eight-character Audit Group code)

Each input card image can contain up to six parameters. Each parameter specifies an Audit Group, or 'KEYAUDIT' or 'COMMON00', for which a rule is to be created. An Audit Group consists of eight characters, a four-character Application System ID ssss and a four-character Audit Group code yyyy. Since the standard naming convention for Key audits is 'KEYAUDIT', the first eight characters of the root key, they must all be grouped within one static rule. Key audits, with the alternate naming convention, can be grouped with other audits using the same application system ID and audit group code. Also, all Common Audits must be combined in one static rule named 'COMMON00'. An Audit Group is associated with one or more transactions within an Application System ID. This means that each static rule created by the program will have all the audit descriptors needed to audit all fields within the transaction(s) with the exception of Key and Common audits. Since multiple transactions can have the same Audit Group code, a single static rule can service more than one transaction.

Output from the batch program is Assembler Language source statements that can be put in to the Rules Generator to create the compiled and link-edited rule. Output is controlled through the DDNAME CARDOUT in the procedure. The file is punched to cards (SYSOUT=B), but should be modified to create a sequential file on tape or disk. This file should be presented as SYSIN to the Rules Generator. A listing also is printed that shows all output and any error messages issued by the program.

Please note that the length from the beginning of the first leg to the beginning of the last leg within the static audit rule may not be greater than 64K.

CHAPTER 6. SPECIAL PROCESSING AND USER EXITS

Many applications require unique processing functions in addition to the functions supplied by the common modules. IMSADF II is structured such that unique application-dependent code can be easily included in the system. Special processing routines can make use of functions provided by the common modules, such as auditing, data base I/O, message sending, secondary transaction creation, and data mapping. This ability to combine common module functions with application-dependent code can simplify the task of coding and debugging an application program.

Special processing routines make full use of specific common modules through CALL statements. The special processing routines can access the Scratch Pad Area (SPA), which contains preloaded rules and segments for a transaction. Since special processing routines have access to Input Transaction and Segment Layout Rules via the SPA and Audit Rules by means of the Auditor module, segment and field layout changes need not affect the logic of programs.

On entry, the transaction driver checks the Input Transaction Rule to determine if any segments should be loaded. If so, the segments and Segment Layout Rules are loaded into the SPA. The Input Transaction Rule is checked again to determine whether control should be given to the special processing routine or a screen should be displayed. If screen display is bypassed, the transaction driver immediately passes control to the special processing routine for pre-display transaction processing. If screen display is specified, the transaction driver formats and displays a screen using the segments just loaded and the Formatter module. Once the displayed screen has been changed and entered by the user, the transaction driver is re-entered and the input is mapped back to the segment area of the SPA. The transaction driver then passes control to the special processing routine. Once in control, this routine can perform calculations specific to the application, invoke specific common modules, or call other user-written routines. When the special processing routine is completed, it returns control and a return code to the transaction driver. The return code specifies to the transaction driver the next action to be taken.

The application designer is responsible for selecting the type of IMSADF II processing that best fits the transaction. In order to do this, a number of factors must be considered. Among these are:

- Data base requirements
- IMSADF II processing facilities
- Unique application requirements not provided by IMSADF II
- Input/Output requirements

Once the application requirements have been established, one of the following processing techniques can be chosen:

- Standard Processing
- Special Processing

The more use an installation makes of Standard Processing, the less it will need Special Processing. However, most application systems ultimately will require some Special Processing. The programs written to include additional logic with IMSADF II are referred to as Special Processing Routines (SPR). These routines can be combined with the functions provided with Standard Processing to enhance the power and flexibility of IMSADF II

An SPR will be required for:

- Generating reports
- Processing twin occurrences of a segment

- Unique processing logic required by the application that cannot be handled through the auditor

Special Processing should be considered an extension of the functions provided under Standard Processing. This means that conversational functions such as sign-on validation, Primary and Secondary Option Menu selections, and Key Selection are as available to an SPR as to other conversational applications. In addition, the user has the option of using the Conversational, Nonconversational, and Batch Transaction Drivers to load rules and segments prior to calling the SPR needed to provide unique logic.

SPECIAL PROCESSING ROUTINES (SPRS)

Special Processing Routines receive control through a call by a transaction driver (Conversational, Nonconversational, or Batch). Once in control, the routine can:

- Retrieve or update segments by calling the appropriate Segment Handler (Rule).
- Map data between the Scratch Pad Area (SPA) and a work area in an SPR. Data from any fields located in segments currently in the SPA can be moved back and forth between the SPA and the SPR work area. This transfer is handled through a call to the Mapping Routine using the appropriate mapping rule.
- Validate data that has been changed through input data by calling the Auditor module.
- Build and display screens (in addition to the Standard display screen) by calling the Terminal Message Writer module.
- Perform application-dependent logic and calculations.

Before calling the SPR, the transaction driver will load into the SPA:

- The Input Transaction Rule specified by the Special Processing Transaction ID.
- Any data base segments required from the hierarchical path(s) as described by the DBPATH Rules Generator parameter. It will also allocate space for other data base and pseudo segments. The Segment Layout Rules that have been identified in the ITR for any segment (i.e., data base segments and pseudo segments).

Once loading is complete, in Conversational mode, the transaction driver will act according to the BYPASS specification in the ITR and will either call the SPR (BYPASS=YES) or will format and display any pseudo segments and segments that are defined in the SPA (BYPASS=NO). The terminal user then can enter or modify fields on the screen displayed, and, when the screen is entered, the transaction driver will call the SPR.

When operating in Nonconversational mode, the SPR will be called immediately after the input data is read. When operating in Batch Mode, BYPASS=YES causes the transaction driver to inhibit the mapping of any data prior to calling the SPR. The BYPASS=YES specification should be used if the transaction invokes an SPR that does not require input data or if additional segments are required before the input data is mapped into the segment fields.

The input data read by the transaction driver is mapped into segment fields in the SPA according to specifications in the Input Transaction Rule. This means that if the SPR gains control prior to the reading of input data (BYPASS=YES), the segment fields will not reflect any updates. If the SPR gains control after the input data has been read (BYPASS=NO), updates to segment fields will be reflected.

SPECIAL PROCESSING INTERFACE ROUTINE

Special Processing Routines can be written in PL1 (non-MAIN procedures with the Assembler option on the interface routine entry points), COBOL, or Assembler Language. A Special Processing Interface Routine (SPIR) has been provided to help the programmer to interface with the transaction driver and common modules. The Special Processing Interface Routine performs the following functions:

- Provide address calculations and manipulations that are difficult to handle in high level languages.
- Allow the user to concentrate on non-IMSADF II data elements.
- Keep track of the location of a segment in the SPA.
- Keep track of the current (last) key for each segment.
- Allow the use of DL/I-like mnemonics to indicate which function the common segment handler is to perform.

The transaction driver selected by the programmer calls an SPR through the SPIR. Depending upon the language in which the SPR is written, the proper linkage is established and the parameters are passed. All calls to common modules by the SPR are passed through entry points in the SPIR. The SPIR determines which rules are needed and the SPA areas to be accessed, and it passes the appropriate parameters to the common modules. Figure 6-1 illustrates how the SPIR interfaces with an SPR.

Note: The SPIR expects any user parameter list to be a variable list, hence, the list end indicator (X'80') must be turned on in the last parameter.

The functions of the SPIR are initiated by subroutine calls in the user's SPR. These functions fall into five general categories which are:

1. IMSADF II's Common Module Invocation routines
2. Segment Key Manipulation routines
3. Field Control Routines
4. Extended Data Base Call Setup routines
5. Automatic Data Base Updating routines

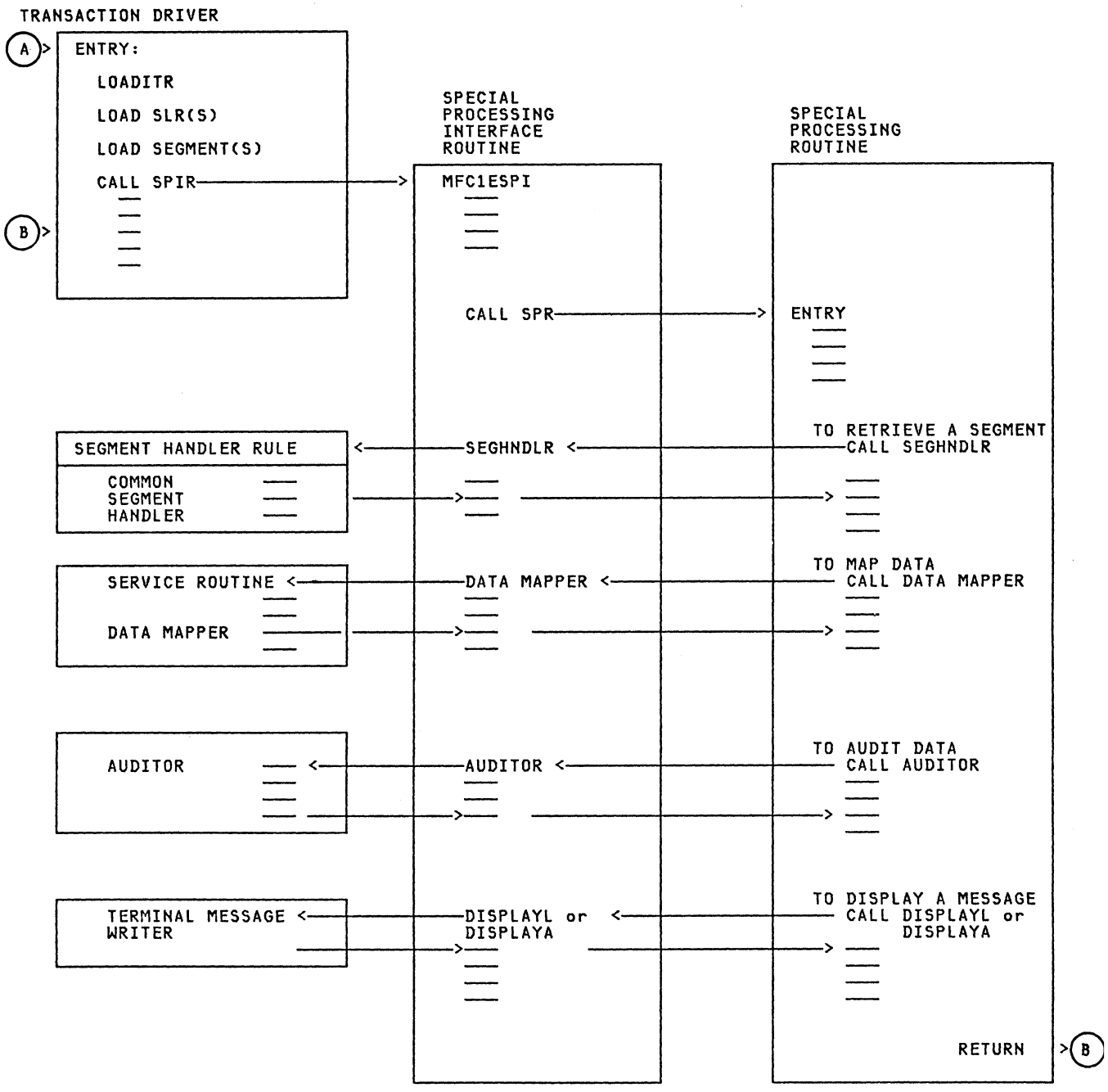


Figure 6-1. Special Processing Interface Routine

DESIGNING A SPECIAL PROCESSING ROUTINE

When designing an SPR, the programmer should keep in mind those functions that can be handled by one of the transaction drivers or the common modules. By incorporating these functions into the SPR, programming and checkout effort will be reduced and data independence will be enhanced.

The following are brief descriptions of cases in which an SPR might be used. In each case, the logic the programmer might use is discussed. Detailed call and return parameters are discussed later in the chapter.

Case 1

Requirement: Display data from segments in a hierarchical path. Allow the terminal operator to modify fields. Validate modifications, compute new values from modified fields, and update the data base if the updates are correct.

Logic: The segments are loaded and displayed by a transaction driver (BYPASS=NO). When the terminal operator enters data, the SPR is called. The SPR calls the Auditor to validate the data that has been entered. If the Auditor detects an error, control is returned to the transaction driver with the specification that the screen is to be redisplayed with appropriate error messages. If there are no errors in the validation, the appropriate field to be computed can be moved from the SPA to the work area by calling the Data Mapper. Once the calculations have been performed, the data can be moved back to the SPA by recalling the Data Mapper. The segments then can be updated in the data base by calling the appropriate Segment Handler Rules. In writing all of the above logic, the application programmer is not concerned with the segment layouts, audit criteria, or data base structures. The only requirement is that the programmer must know the IDs of the segments on which he is operating.

Note: This case could also be handled with Standard Processing, combined with the arithmetic and logic rules of the Auditor.

Case 2

Requirement: Retrieve all the twin occurrences of a segment type, accumulate fields, and display the totals.

Logic: The parent segment of the twin occurrences is retrieved by the appropriate transaction driver and the SPR is called. The first twin segment is retrieved by the SPR through a call to the Segment Handler. The SPR moves the appropriate fields from the SPA to the work area in the SPR by calling the Data Mapper. The fields are added to counters in the SPR and the next segment is retrieved by calling the Segment Handler again. This process is iterated until no more segments are found under the parent segment. The SPR accumulation counters are then moved to the SPA through a call to the Data Mapper, after which the SPR returns to the transaction driver with the specification that the screen is to be displayed.

Case 3

Requirement: Process data entered from the terminal and display an information screen back to the terminal. Once the terminal user has read the information provided and has signaled the system by pressing the ENTER key, redisplay the original screen for additional entry.

Logic: The segments are loaded and displayed by the appropriate transaction driver. When the terminal operator enters data, the SPR is called. The SPR performs the appropriate calculations and produces message data. The message produced from this data is handled line by line and can be

included in up to fifteen logical screens. The SPR calls the Terminal Message Writer to format and display the message. The SPR then returns to the transaction driver with the specification that a screen has been displayed and that when that screen is reentered, the transaction driver is to display the original screen. After reading the informational display, the terminal user presses the ENTER key, which causes the transaction driver to redisplay the updated segment display screen.

It should be noted that, although the programmer is designing and coding an application module, IMSADF II rules can make up a major portion of that module's logic. The Input Transaction Rule controls which segments a transaction driver loads and the sequence in which the SPR is called. The mapping rule controls which fields are moved between the SPA and the SPR work area. The Segment Handler Rules control the actual access of the data base segments. The Audit Rules control the validation routines and error messages associated with a particular field.

CODING A SPECIAL PROCESSING ROUTINE

When coding an SPR, the programmer should be familiar with some of the fields located in the SPA and how they are used. Nonconversational and batch processing have a simulated SPA area that corresponds to the conversational processing SPA. A data definition (dummy section) for the SPA area is provided on the distribution tape in the IMSADF.ADFMAC library for each of the programming languages (PL/I, COBOL, BAL). In addition, Appendix B in the IMS Application Development Facility II Version 2 Release 2 Diagnosis Guide contains a detailed description of this area.

Some of the SPA fields of general importance to the SPR programmer are:

- SPAFIRST (Half Word Binary) - set to zero by a transaction driver on the initial call to the SPR for a transaction. Since the SPR iterates with the driver any number of times depending upon the return code, this field can be used to keep track of which iteration is being processed. The batch transaction driver also sets SPAFIRST to (-1) on the End-of-File call and to (-2) on a Restart call.

- SPARTNCD (Full Work Binary) - contains the return code from the SPIR.

- SPADLIST (Character 2) - contains the DL/I status code returned from a Segment Handler call.

- SPAERMSG (Character 50) - can be used to display a programmer generated message on the message line of the segment display screen in Conversational or Nonconversational processing. A programmer can place a message in this field and return to the appropriate transaction driver with a return code of 3. This causes the segment display screen to be displayed with the message. When the Batch Transaction Driver is in control, the message will be printed on the system printer. When the Nonconversational Transaction Driver is operating with a no-response Input Transaction Rule, the message will be printed on a 3284/3286 printer.

- SPAKEYID (Character 255) - contains the fully concatenated key of each target segment path that was built during key selection or that was entered with the transaction in Nonconversational or Batch processing. The programmer may want to use a portion of this key to access segments in a different hierarchical path under the same root segment.

- SPASECTX (Halfword Binary) - indicates that a secondary transaction should be generated by the conversational or nonconversational transaction driver. A value of 1 indicates a secondary transaction will be generated for each STX operand specifying OK. A value of 2 indicates a secondary transaction will be generated for each STX operand equal to ER. A value of 3 will generate a secondary transaction when STX=OK or ER. If the Special Processing Routine calls the Auditor and the Auditor indicates secondary transactions

are to be generated, a value of 1, 2, or 3 in SPASECTX will cause the Driver to initiate the secondary transaction.

- 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. 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 (R.C.=5) will cause the program to process as if the terminal user had modified the TRX: field on the Segment Display Screen.

- SPAUTILY and SPAFLDSG (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.

- SPACOMLN (Halfword Binary) - The length of the area addressed by SPAUTILY and SPAFLDSG. If this field contains binary zeroes, it means that no programmer maintained area exists in the SPA.

In addition to the SPA fields, the programmer should be aware of the manner in which a transaction driver uses its option to load segments prior to calling the SPR. The Input Transaction Rule describes which segments are to be loaded or which segment areas are to be reserved in the SPA. A maximum of one hundred segments, segment areas, or a combination of segments and segment areas can be loaded into the SPA. As a segment is loaded from a data base, a segment table is constructed that points to the segment and its Segment Layout Rule. The fully concatenated key for the segment is also saved in the SPA. The discussion of the Segment Handler Call that is presented later in this chapter will explain how this key can be used to assist the programmer in the data base accesses. Two additional types of segment areas also can be reserved. The first area contains space for a segment that the SPR is to retrieve. When a segment is to be retrieved by an SPR, an entry is placed in the segment table and the Segment Layout Rule is loaded into the SPA. The second area contains space for a pseudo segment. Pseudo segment areas are reserved for fields that are not in data base segments. They can be thought of as work area space to hold screen input, transaction input, or SPR computational fields that do not map directly into data base segments. Additionally, the segment table points to the pseudo segment area, and a Segment Layout Rule is loaded to the SPA for the pseudo segment. Additional information concerning pseudo segments is presented later in this chapter.

The restrictions on the combination of segments and segment reserve areas are:

- Maximum of 100 segments and segment reserve areas per transaction

- Segments within each hierarchical path - 1 to 15

TRANSACTION DRIVER INTERFACE

When a transaction driver calls an SPR, the following parameters are passed:

```
  - Addr     SPA
  - Addr     Segment Handler Communication Area
  - Addr     Audit Data Base PCB
  - Addr     Message Data Base PCB
  - Addr     1 to 120 Application Data Base PCBs
 *- Addr     Logical Terminal PCB
 *- Addr     Alternate Terminal PCB
 *- Addr     Express Terminal PCB
```

* The Batch Transaction Driver does not pass these addresses to the SPR in a DL/I batch environment. For the Batch Transaction Driver operating in a Batch Message Processing Region and accessing a Program Specification Block (PSB) with these I/O PCBs defined, the addresses will be passed.

In most cases, the only parameter that should concern the application programmer is the address of the SPA. All other parameters are known and controlled by the SPIR for subsequent common module calls.

Return Codes

Return codes are issued by the SPR when control is returned to the transaction driver. These return codes control the initial processing performed by the driver after the return and the action taken by the driver the next time it is reentered. The return codes that are issued and the meaning of each code will differ depending upon which transaction driver is being used. The following three lists show the return codes and their meanings for each transaction driver.

CONVERSATIONAL

R.C.	Initial Transaction Driver Action	Reentry Transaction Driver Action
0	Return to Primary Option Menu	New Transaction.
1	Generate Secondary Transactions as required according to the setting of SPASECTX. Criteria for generating the Secondary Transaction is in the Input Transaction Rule, the setting of SPASECTX and the field auditing. Control is immediately returned to the Special Processing Routine.	Does not apply. There is no screen interaction with the end-user.
2	Display Screen.	Return to SPR.
3	Display Screen with message in 'SPAERMSG'.	Return to SPR.
4	Display Screen with message 'SPECIAL PROCESSING SUCCESS-FULLY EXECUTED'.	Return to SPR.
5	The transaction ID named in SPACGTRX is set up as the next transaction ID to be processed.	Display screen for the next transaction ID.
8	Display Screen with Message 'ENTER E TO DISPLAY ERROR MSG'.	Display Error Message or return to SPR.
12	SPR invoked Message Display. Terminate.	Display Screen.
24	Issue ROLLCALL to back out any updates done by this transaction.	Does not apply. Transaction will have abended.
28	Generate the error message returned from Segment Handler and terminate. (See Note 1.)	Return to Primary Option Menu.
XX	SPR invoked message Display. Terminate (XX=Other).	Return to Primary Option Menu.

Notes:

1. The error message is specified by a message number in the 'COMSG' field of the Segment Handler Communication area.

2. Secondary Transaction Generation is invoked for any of the above return codes if SPASECTX is set to a non-zero value.

Nonconversational and Batch processing do not reenter the driver in the same manner as Conversational processing. The interaction with the screen is not provided. This is the reason the return codes have different meanings or manners in which they are handled.

NONCONVERSATIONAL

R.C.	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 terminal printer) the message in 'SPAERMSG' if this is a no-response 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 transaction.
4	Print (to terminal printer) 'Special Processing Successfully Completed' if this is a no-response Input Transaction Rule. Display the message if this is a response Input Transaction Rule according to Output Format Rule specifications. Read the next transaction.
5	The transaction ID named in SPACGTRX is set up as the next transaction ID to be processed. The Segment 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 terminal printer) or display audit error message and read the next transaction.
12	SPR generated a message. Read the next transaction.
24	Issue ROLLCALL to back out any updates. Pseudo abend caused.
28	Generate and print (to terminal printer) or display the error message returned from the Segment Handler and read the next transaction. **Note:** The error message is specified by a message number in the 'COMSG' field of the Segment Handler Communication area.
xx	Read the next transaction. (xx=other)

Note: Secondary Transaction Generation is invoked for any of the above return codes if SPASECTX is set to a non-zero value.

BATCH

R.C.	Transaction Driver Action
0,2	Read next transaction. **Note:** Included for compatibility with Conversational and Nonconversational Processing modes.
1	Generate Secondary Transactions as required according to the setting of SPASECTX. Control is immediately returned to the Special Processing Routine.
3	Output message in 'SPAERMSG' and read next transaction.
4	Output 'SPECIAL PROCESSING COMPLETED SUCCESSFULLY' and read next transaction.
8	Output audit error message and read next transaction.
12	SPR wrote error message. Read next transaction.
24	Issue ROLLCALL to back out any updates. Pseudo abend caused. **Note:** In batch the user must use the Data Base Back Out Utility of IMS/VS to back out the updates.
28	Generate and print error message returned from the Segment Handler. Read the next transaction.
xx	(xx=other). Invalid return code. Message issued to this effect.

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.

Additional Functions

A transaction driver performs two additional functions upon return from an SPR.

- First, it checks to determine if automatic message sending is required. This check is a test of the flag SPAMSUTL in the SPA. This flag is set by the SPIR if a return from the Auditor specifies that messages are to be sent.

- Next the setting of the flag SPASECTX is checked to determine if a secondary transaction should be generated. The generation of a secondary transaction is controlled by specifications in the Input Transaction Rule, the Audit Data Base, and the Message Data Base.

- Upon completion of these functions, the transaction driver continues processing as described under the specific return codes.

COMMON MODULE INTERFACES

An SPR can use the functions provided by the common modules that are called by the SPR through individual entry points in the SPIR. The common module entry points are:

AUDITOR	-	Interfaces with the Auditor module for audit/logic processing
MAPPER	-	Interfaces with the Data Mapper for moving data
COPYSEG	-	Interfaces with Data Mapper to copy a segment to or from the SPA
SEGHNDLR	-	Interfaces with the Common Segment Handler for data base I/O
SEGINIT	-	Initializes the segment I/O area in the SPA
DISPLAYL DISPLAYA DISPLAYE	-	Interfaces with the Terminal Message Writer for Message sending to the IOPCB, alternate, or express PCB
GETKEY SETKEY ZEROKEY GETKINFO	-	Manipulates key information for a segment in the SPA
DISPLAYP	-	Interfaces with the Printer (3284-3286) Terminal Message Writer for sending a message to a remote printer
SETATTR		Sets Message Format Service attributes for a field displayed on the screen
SETCOLOR SETXHILT	-	Sets Message Format Service color or extended highlighting attributes for a field displayed on the screen
SETCHANG	-	Modifies the field changed flag
SETERROR	-	Sets a field in error and identifies the error message number
SETCC SETPATH SETSSA SETUNQ RSETSEGH	-	Provides extended data base call setup functions
SEGUPDTE	-	Interfaces with the data base update module to update, delete or insert any segment requiring update in the SPA
SETFLAG	-	Modifies the segment delete, retrieve or change flag in the SPA

The following is a discussion of the usage and call sequence of each of these modules.

AUDITOR

The SPR should call the Auditor whenever a validation of segment fields or input data is needed. Key Audit and Preaudit processing through the transaction driver occurs prior to the SPR being invoked. When Special Processing is being used, the Standard Audit call is made by the SPR rather than the transaction driver. The fields that the Auditor is to validate are located in data base or pseudo segments residing in the SPA area at the time of the call. One call to the auditor validates <u>all</u> fields specified as requiring field auditing, automatic field assignment, or automatic message sending.

The call format to invoke the Audit is: CALL AUDITOR.

When control is returned from the Auditor, the SPR should check the return code, which is located in SPARTNCD. The codes that can be returned are:

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 determined that messages must be sent as a result of specific field updates. The SPIR will turn on a flag in the SPA (SPAMSUTL), which 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 a return code of '8' to display the Warning Messages. If the terminal user enters a 'U' on the message screen, the SPR is recalled with a 99 in SPARTNCD. The SPR can then perform the appropriate updates. If the terminal user does not enter a '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 and has set the appropriate error message number(s) in the Segment Layout Rule(s). The SPR should return control to the transaction driver at this point with return code '8' to display or log the error message.

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 this point with return code '8' to display or log the error message.

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.

In the Case 1 example discussed earlier, the Auditor is called by the SPR upon initial entry to the routine because data already has been entered by the terminal user. For return codes 8, 12, and 16, the SPR should return immediately to the transaction driver. Otherwise, processing continues.

MAPPER

The SPR should call MAPPER when fields are to be mapped between the SPA and the SPR work area. Data can be mapped to or from fields in different segments in the SPA by a single call to the Data Mapper. Figure 6-2 illustrates the mapping of fields from different segments in the SPA to the SPR work area.

MAPPER uses a mapping rule to determine which fields are to be mapped from or to the SPA and in what order the mapping will occur. The mapping rule is created by the Rules Generator and is a Segment Layout Rule for an area in the SPR.

The call format to invoke the MAPPER is:

```
CALL MAPPER(ID,MAPAREA,MAPCODE);            (in PL/I)

CALL 'MAPPER' USING ID, MAPAREA, MAPCODE.   (in COBOL)
```

ID = the two-character segment ID of the mapping rule

MAPAREA = area in the SPR where the data will be mapped to or from

MAPCODE = fullword code that specifies the direction of data movement

0 = MAP from the SPA to the SPR
1 = MAP from the SPR to the SPA

During the mapping operation, data conversion may take place if the field type differs between the Segment Layout Rule and the mapping rule. Valid data conversions are covered in the IMS Application Development Facility II Version 2 Release 2 Application Development Guide.

MAPPER returns the following return codes in SPARTNCD:

0 - Normal Completion.

4 - Conversion Error. Return to the transaction driver with return code '8' to display or log the error message.

The Case 1 and Case 2 examples both have a situation in which mapping could be used. In Case 1, the same mapping rule could be used to map the data to the SPR and back to the SPA. The only change would be MAPCODE=0 or 1. In Case 2, two mapping rules might be used, one for moving the individual fields to the SPR, another for moving the accumulated counters back to the SPA.

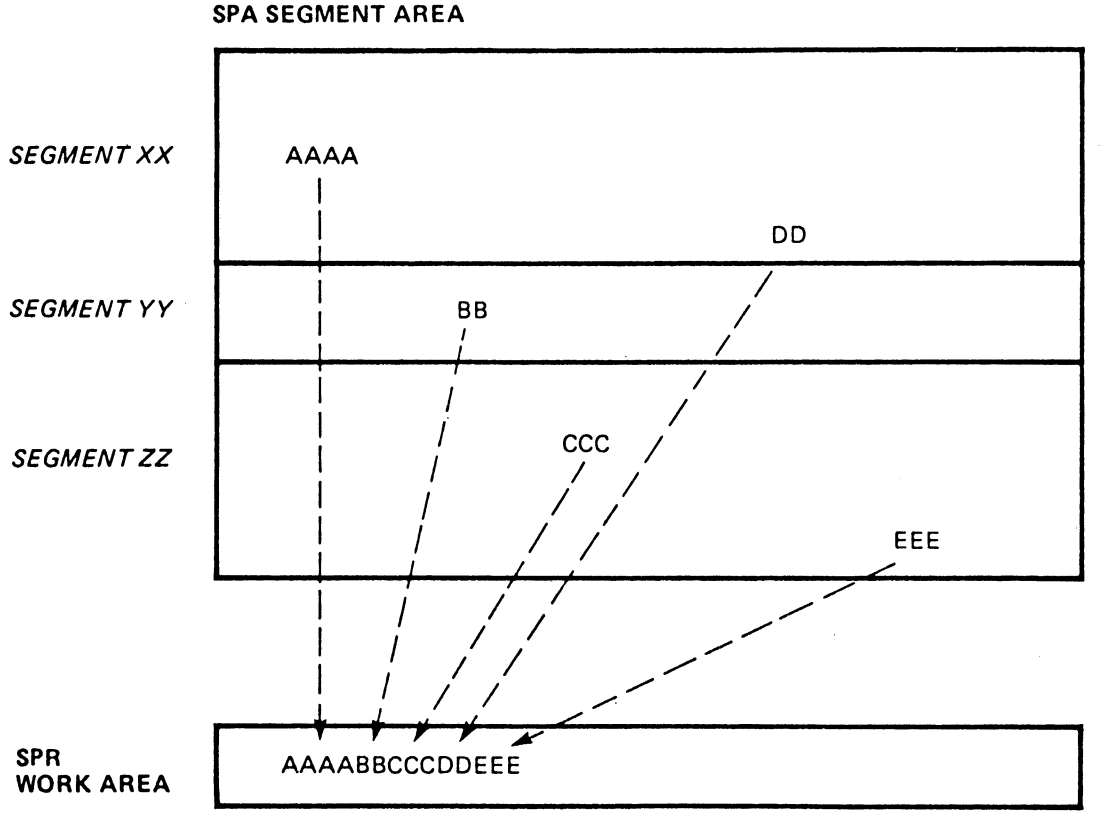


Figure 6-2. Mapping Fields Between the SPA Segment Area and the SPR Work Area

COPYSEG

The SPR should call COPYSEG when a segment of data is to be copied between the SPA and the SPR work area without consideration for fields.

The call format to invoke COPYSEG is:

```
CALL COPYSEG(ID, SPRAREA, COPYCODE);     (in PL/I)

CALL 'COPYSEG' USING ID, SPRAREA, COPYCODE.  (IN COBOL)
```

ID =	the two-character segment ID of the segment to be copied.
SPRAREA =	area in the SPR that the data will be copied to or copied from.
COPYCODE =	fullword code that specifies the direction of data movement: 0 from SPA to the SPR 1 from the SPR to the SPA.

COPYSEG returns the following return codes in SPARTNCD:

0 - Normal Completion

4 - Invalid Segment ID

SEGHNDLR (SEGMENT HANDLER)

The SPR can retrieve or update data base segments through calls to SEGHNDLR. This function, in conjunction with the appropriate Segment Handler Rule, provides all the I/O function needed to retrieve, add, update, and delete segments. The use of the Segment Handler eliminates the need for programming detailed DL/I specifications such as Segment Search Arguments (SSA). Each segment type that is accessed under the IMSADF II will have a Segment Handler Rule that contains the SSA specifications needed.

Note: VSAM files are accessed as though they are root-only DL/I data bases. SPR can use the SEGHNDLR interface to retrieve or update VSAM files.

The SPR is responsible for all data base updates (replace, insert, or delete). This can be accomplished either by individual SEGHNDLR calls or by the Automatic Data Base Update (SEGUPDTE) facility. Only the SEGUPDTE facility will verify that segments to be updated (with the COMPARE=YES attribute specified during rules generation) have not changed since they were originally retrieved. The logic of the routines should determine when and how a segment should be updated and make the appropriate Segment Handler call. In addition, any retrieval of segments other than the segments loaded by the transaction driver must be handled in the SPR.

The programmer can perform data base I/O either into an area in the SPR or into a segment area reserved in the SPA. If the SPA area is used, the segments accessed must be described by the Input Transaction Rule. I/O performed in the SPA has a number of advantages:

- Segment Handler call parameters are simplified.
- The system maintains the current key and updates it automatically upon a successful retrieval.
- The Auditor can be invoked to validate fields in the SPA.
- Input data is mapped automatically to the appropriate segment field(s) by the transaction driver.

- Segment fields in the SPA can be formatted automatically and displayed by the transaction driver.

- The programmer can retrieve the specific field(s) required by calling the Data Mapper for segments maintained in the SPA area. This means that access to data is controlled at the field level through specifications in the mapping rule.

The call format for the Segment Handler is:

```
CALL SEGHNDLR(ID,FUNC,KEY,OPER,PCB#,AREA)              (in PL/I)

CALL 'SEGHNDLR' USING ID, FUNC, KEY, OPER, PCBNO. AREA.(in COBOL)
```

The first two parameters of the CALL are required. The remaining four are optional depending upon the access requirements.

ID = two-character target segment ID. This ID is used to identify the appropriate Segment Handler Rule and to determine whether a segment area in the SPA is needed.

FUNC = Four-character mnemonic for the DL/I function(s) to be performed. The allowable parameters are:

'GU '	GET UNIQUE
'GUU '	GET UNIQUE - target SSA unqualified
'GHU '	GET HOLD UNIQUE
'GHUU'	GET HOLD UNIQUE - target SSA unqualified
'GN '	GET NEXT
'GNQ '	GET NEXT - target SSA qualified
'GHN '	GET HOLD NEXT
'GHNQ'	GET HOLD NEXT - target SSA qualified
'GNP '	GET NEXT WITHIN PARENT
'GNPQ'	GET NEXT WITHIN PARENT - target SSA qualified
'GHNP'	GET HOLD NEXT WITHIN PARENT
'HNPQ'	GET HOLD NEXT WITHIN PARENT - target SSA qualified
'DLET'	DELETE
'ISRT'	INSERT
'REPL'	REPLACE
'HDEL'	GET HOLD UNIQUE-DELETE*
'HREP'	GET HOLD UNIQUE-REPLACE*

*GET HOLD UNIQUE uses an I/O area in the Segment Handler module. This I/O area is not the user area.

KEY = A field in the SPR that will be used instead of or in conjunction with the 'current key' for this data base call. The 'current key' is the key associated with the last successful data base call to this segment ID. If OPER indicates a full key, this field will be used instead of the 'current key'. If OPER indicates a partial key, the parents' key(s) in the 'current key' will be used and this field supplies only this segment's part (partial) of the fully concatenated key. If the data base call is successful, and the call function is not 'ISRT', both the 'current key' and this field will be updated to reflect the new 'current key' (partial in = partial out for this field). The 'current key' field in the SPA is always updated whether this parameter is supplied or not. If this operand is not specified, the system will use the 'current key' which is kept in a reserved area in the SPA. This key is located using the ID operand. If an occurrence of this segment has not been retrieved previously either by the transaction driver or a previous call by the SPR, a fully concatenated KEY parameter must be specified.

OPER = Specifies the key operator to be used. The allowable parameters are:

'FE'	Full Key - Search Equal (Default)
'FG'	Full Key - Search Greater Than
'PE'	Partial Key - Search Equal
'PG'	Partial Key - Search Greater Than

If FE or FG is specified, the KEY parameter will be treated as the fully concatenated key that is to be used in this data base call. For PE or PG, the KEY parameter supplies only this segment's part (Partial part of) of the key is concatenated key. The parent's portion of the key is obtained from the 'current key' area in the SPA. Use of the Partial Key parameter requires that the segment be described in the Input Transaction Rule.

PCB# = Fullword that specifies the number of the user PCB that is being used. This number can be from 1 through 120. If this operand is not used, the default is the PCB number in the Input Transaction Rule of the segment specified by the ID operand.

AREA = Specifies an area in the SPR in which data base I/O is to be performed. If this operand is not used, the default is the segment area in the SPA that is specified by the ID operand.

When any of these six operands is specified with the SEGHNDLR call, all operands that precede it also must be specified. If a segment is not described in the Input Transaction Rule, then all data base I/O must be performed in the SPR area. When I/O is performed in the SPR area all six operands must be specified.

If a segment is specified in the Input Transaction Rule, each SEGHNDLR call that specifies the KEY operand will cause the given key to be placed in the segment key area in the SPA. A segment specified in the Input Transaction Rule also results in each successful retrieval call (GN,GHN,GNP,GHNP) specified with the KEY operand to cause the updated, fully-concatenated key to be returned in both the user key area of the SPR and the segment key area of the SPA. The segment key area of the SPA is separate from the actual segment area in the SPA.

In the Case 3 example, the first twin occurrence will be retrieved with the following call sequence.

```
CALL SEGHNDLR(ID,'GNP ',KEY);              (in PL/I)

CALL 'SEGHNDLR' USING ID, GNP, KEY.        (in COBOL)
```

The 'SPAKEYID' field will contain the fully concatenated key of the parent and can be used as the key field in this call. All subsequent calls will be:

```
CALL SEGHNDLR(ID,'GNP ',KEY);          (in PL/I)

CALL 'SEGHNDLR' USING ID, GNP.         (in COBOL)
```

Each call will place the segment in the SPA area specified by the ID operand.

Two additional functions can be given in a SEGHNDLR call. The format of the call is altered to the following:

(Use this form after SETSSA)

```
CALL SEGHNDLR(ID,FUNC,PCB#[,AREA]);             (in PL/I)

CALL 'SEGHNDLR' USING ID, FUNC, PCBNO[, AREA].  (in COBOL)
```

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 function or for the SGN function if the ID is blank (' '). When the call completes successfully, the Input Transaction Rule (ITR) is scanned to see if the retrieved segment is defined in it. If so, a blank (' ') ID will be changed to the segment ID of the retrieved segment, while a non-blank ID will remain unchanged.

For the SGN function, if the ID is not blank and the segment is defined in the ITR, 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.

A 'GA' or 'GK' status code 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.

0 - DL/I Call Successfully Completed.

4 - Non-blank 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.

16 - Non-blank status code from IMS/VS on CHNG status is in SPADLIST.

SQLHNDLR (SQL HANDLER)

The SPR can retrieve or update DB2 data base tables through calls to SQLHNDLR. This function, in conjunction with the appropriate Table Handler Rule, provides the I/O function needed to select, insert, update, and delete tables.

The SPR is responsible for all data base updates (update, insert, or delete). This can be accomplished either by individual SQLHNDLR calls or by the Automatic Data Base Update (SEGUPDTE) facility. Only the SEGUPDTE facility will verify that tables to be updated (with the DATACOMP attribute specified during rules generation) have not changed since they were originally retrieved. The logic of the routines should determine when and how a table should be updated and make the appropriate Table Handler call. Selects on tables other than the tables loaded by the transaction driver must also be handled by the SPR.

The programmer can perform data base I/O either into an area in the SPR or into a table area reserved in the SPA. If the SPA area is used, the tables accessed must be described by the Input Transaction Rule.

The format of the SQLHNDLR call is:

In COBOL

```
WORKING STORAGE SECTION.
77 ID     PICTURE XX.
77 FUNC   PICTURE XXXXXXXX.
77 KEY    PICTURE X(n).  NOTE n is defined as necessary.
77 AREA   PICTURE X(n).  NOTE n is defined as necessary.
77 TLR    PICTURE X(n).  NOTE n is defined as necessary.
EXEC SQL
  INCLUDE SQLCA
END-EXEC

CALL 'SQLHNDLR' USING ID, FUNC, {KEY, AREA, TLR, SQLCA}.
```

In PL/I

```
DCL (SQLHNDLR) ENTRY OPTIONS(ASSEMBLER,INTER);
DCL  ID    CHAR(2),
     FUNC  CHAR(8),
     KEY   CHAR(n),    /* n is defined as necessary  */
     AREA  CHAR(n),    /* n is defined as necessary  */
     TLR   CHAR(n);    /* n is defined as necessary  */
EXEC SQL INCLUDE SQLCA ;

CALL  SQLHNDLR (ID,FUNC,{KEY,AREA,TLR,SQLCA});
```

ID is any valid table/view id defined to IMSADF II.

FUNC is an eight-character standard SQL function label or user supplied label for execution. Valid FUNC values are:

```
SELECT, UPDATE, DELETE, INSERT
CSELECT, CUPDATE, CDELETE
KSELECnO, KSELECnF, KSELECnC
CSELECTO, CSELECTC
CUPDATEO, CUPDATEU, CUPDATEC
CDELETEO, CDELETED, CDELETEC
```

USER SUPPLIED - where label is 1 to 8 characters and is the label specified as the SQLCALL in the generate statement when OPTIONS=TABH and SQLUSER=YES.

KEY, if specified, defines an area in the EXIT that contains the WHERE clause host variables. It must represent a concatenation of the host variables in the WHERE clause and in the order specified in the WHERE clause.

AREA is specified if the exit wants the row returned to its area, otherwise IMSADF II will read it into the SPA.

TLR, if specified, is the Table Layout Rule associated with table ID.

SQLCA, if specified, is used for the SQL call. SQLCA is interrogated for completion status (SQLCODE, SQLWARN0 to 7). If not provided, the driver will use its copy. This address (SQLCA) is one passed to the SPR.

The following return codes are received in SPARTNCD:

0 - SQL was executed. Check SQLCODE for completion status.

4 - SQL was executed. SQLCODE is less than zero or equals one hundred. Check SQLCODE for status.

8 - SQL not executed. FUNC was not in Table Handler Rule for Table (ID).

SEGINIT (INITIALIZE SEGMENT AREA)

The SPR can initialize one or more segment areas with the SEGINIT call.

The call format for SEGINIT is:

```
CALL SEGINIT(ID1,ID2, ...);                (in PL/I)

CALL 'SEGINIT' USING ID1,ID2, ....         (in COBOL)
```

SEGINIT returns the following return codes in SPARTNCD:

0 - All segments initialized.

4 - One or more segments were not found in the SPA.

DISPLAY (TERMINAL MESSAGE WRITER)

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 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 PL/I:

```
CALL DISPLAYL(MSGAREA,HDR,OPTION);
CALL DISPLAYA(MSGAREA,HDR,OPTION,LTERM);
CALL DISPLAYE(MSGAREA,HDR,OPTION);
```

In COBOL:

```
CALL 'DISPLAYL' USING MSGAREA, HDR, OPTION.
CALL 'DISPLAYA' USING MSGAREA, HDR, OPTION, LTERM.
CALL 'DISPLAYE' USING MSGAREA, HDR, OPTION.
```

DISPLAYL = Sends the message to the entering terminal using the IOPCB.

DISPLAYA = Sends the message to an alternate terminal.

DISPLAYE = Sends the message via the EXPRESS alternate PCB. The output destination will be the input LTERM for conversational and response nonconversational transactions. For non-response 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 = 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 previous output.

LTERM = eight-character screen LTERM name of the device to receive the output.

If the message generated is to be displayed via the logical terminal PCB, a return code of 12 or xx (Conversational mode only - xx=other) 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 the ENTER key.

The Terminal Message Writer returns the following codes in SPARTNCD:

0 - Successful completion.

4 - Non-blank status on MSG ISRT. IMS/VS status code is in SPADLIST.

12 - Third input parm does not have last part indicator on. Check format of CALL.

The Case 3 example given above shows a situation in which the SPR calls the Terminal Message Writer to format and display a screen or screens for the terminal user. After the final DISPLAYL call, the SPR can return to the transaction driver with a return code of 12. After the terminal user has seen the generated message, processing can continue with the original screen by pressing the ENTER key.

DISPLAYP (PRINTER TERMINAL MESSAGE WRITER)

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 (specified by OPTION 4 or 5). 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 format is:

```
CALL DISPLAYP(MSGAREA,HDR,OPTION LTERM);            (in PL/I)

CALL 'DISPLAYP' USING MSGAREA, HDR, OPTION, LTERM.   (in COBOL)
```

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, and a sequential page number. If the message header area contains blanks, no header line will be printed.

OPTION - 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.
- 4 = Message text followed by a new page.
- 5 = New page.
- OTHER = Add message text to the output previously received.

LTERM - 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:

0 - Successful completion.

4 - Non-blank status from MSG ISRT. IMS/VS status code is in SPADLIST.

12 - Fourth input parm does not have last parm indicator on. Check format of CALL.

16 - Non-blank status on CHNG call. IMS/VS status code is in SPADLIST. (LTERM name probably doesn't exist in the IMS/VS system.)

SEGMENT KEY MANIPULATION ROUTINES

There are four Segment Key Manipulation routines. These routines provide information about the key of a segment and the PCB number to use for the segment along with the setting, clearing or retrieving of the 'current key' of a segment. The 'current key' is the key associated with the last successful data base call for a segment. Space is reserved in the IMSADF II work area to hold the 'current key' for each segment in that work area (Note that this reserved area is not the field SPAKEYID which is discussed elsewhere in this chapter). The 'current key' is updated in this reserved area and presented to the user when specifically requested in a Segment Key manipulation call. The entry points, and functions of the four routines are:

GETKEY - retrieves the full or partial key of the data base segment from the 'current key' area in the SPA.

SETKEY - sets the full or partial key of the data base segment in the 'current key' area in the SPA.

ZEROKEY - sets the full or partial key in the SPA to binary zeroes.

GETKINFO - retrieves 3 or 4 full words of information:

- OFFSET in concatenated key to key of this segment
- LENGTH for the key of this segment
- PCB# from the ITR for this segment
- PCBADDR is the optional parm to receive the PCB address

The call formats to invoke these routines are:

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]);
```

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].
```

ID = the two-character segment identifier.

KEYA= the area in the user's program where the key is to be moved to or from.

F

P = a character 'F' or 'P' that indicates a Full (default if not supplied) or 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:

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.

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 parameters on a SEGHNDLR call.)

12 - Segment is not a data base segment.

FIELD CONTROL ROUTINES

There are three routines which are called to modify the display attributes, change status, or error status of a field.

SETATTR (Set Field Attributes)

SETATTR is used to set the MFS attributes for a field displayed on the screen. One attribute may be modified per call. The format of the SETATTR call is:

```
CALL    SETATTR(FIELDNAME,FUNCTION,OPTION);       (in PL/I)

CALL    SETATTR USING FIELDNAME,FUNCTION,OPTION.  (in COBOL)

        FIELDNAME =    eight-character name of displayed field
        FUNCTION  =    four-character attribute definition
                       HILT = highlight
                       PROT = protect
                       NDSP = non-display
                       PMOD = premodify
                       CURS = CURSOR
        OPTION    =    one-character ON/OFF option
                       0 = OFF
                       1 = ON
```

Return codes are:

0 - Successful completion.

4 - Invalid field name. The field name specified cannot be found in the Input Transaction Rule.

8 - Attribute conflict. The following conflicts are not allowed:

Set Protect ON/OFF on a protected field (MODE=6)
Set Highlight or Non-display ON/OFF on a non-display field (MODE=7)

12 - Invalid function. The four-character function specified is not one of the five allowed.

SETCOLOR (Set Color Attribute)

SETCOLOR is used to set the color attribute for a field displayed on a 3279 Color Terminal. The format of the SETCOLOR call is:

```
CALL    SETCOLOR(FIELDNAME,COLOR);                    (in PL/I)

CALL    SETCOLOR USING FIELDNAME,COLOR.               (in COBOL)

        FIELDNAME = eight-character name of displayed field
            COLOR = Name of color attribute.  The valid colors are
                    PINK,BLUE,GREEN,RED,WHITE,YELLOW,TURQUOISE
```

Return codes are:

0 - Successful completion
4 - Invalid field name. The field name specified cannot be found in the Input Transaction Rule.
8 - The Input Transaction Rule has not been generated for a color terminal. The field entry is too small to contain the color attribute.

SETXHILT (Set Extended Highlight Attribute)

SETXHILT is used to set the extended highlight attribute for a field displayed on a 3279 Color Terminal. The format of the SETXHILT call is:

```
CALL    SETXHILT(FIELDNAME,XHILT);                    (in PL/I)

CALL    SETXHILT USING FIELDNAME,XHLIT.               (in COBOL)

        FIELDNAME = eight-character name of displayed field
            XHILT = Name of the extended highlight feature.
                    UNDERSCORE - Underscore the field.
                       REVERSE - Set field to REVERSE VIDEO.
                         BLINK - Cause field to blink.
                       DEFAULT - Reset highlight to no value.
```

Return codes are:

0 - Successful completion.
4 - Invalid field name. The field name specified cannot be found in the Input Transaction Rule.
8 - The Input Transaction Rule has not been generated for a color terminal. The field entry is too small to contain the extended highlight attribute.

SETCHANG (Set Change Status)

SETCHANG is used to modify the Change flag of a field. Since the Change flag in conjunction with AUDIT=Y or MSG=Y is used by the Auditor to determine if a field should be audited, this call can be used to control field auditing. The format of the SETCHANG call is:

```
CALL     SETCHANG(FIELDNAME,OPTION);          (in PL/I)

CALL     SETCHANG USING FIELDNAME,OPTION.     (in COBOL)

         FIELDNAME =    eight-character name of field
         OPTION    =    one-character ON/OFF option

                        0 = OFF
                        1 = ON
```

Return codes are:

0 - Successful completion.
4 - Invalid field name. The field name specified cannot be found in a Segment Layout Rule.

SETERROR (Set Error Status)

SETERROR is used to allow the Special Processing routine to mark a field in error and specify an appropriate error message number. This call may be used in conjunction with an Auditor call. If the SETERROR call is issued prior to the AUDITOR call, the Auditor will treat any fields marked in error as if the Auditor had found them. This will terminate any Auditor processing in the message leg. The format of the SETERROR call is:

```
CALL     SETERROR (FIELDNAME,ERROR#);          (in PL/I)

CALL     SETERROR USING FIELDNAME, ERROR#.     (in COBOL)

         FIELDNAME =    eight-character name of the field.
         ERROR#    =    Halfword error message number.
                        (1-9999)
```

Return codes are:

0 - Successful completion.

4 - Invalid field name. The field name specified cannot be found in a Segment Layout Rule.

SETFLDOL (Set Field Outlining Attributes)

The SPR can set field outlining attributes on a display field through calls to SETFLDOL. This function may not be valid for non-DBCS terminals.

The format of the SETFLDOL call is as follows:

```
In COBOL
          CALL 'SETFLDOL' USING FIELDNAM, OUTLIN.
In PL/I
          CALL  SETFLDOL (FIELDNAM,OUTLIN);
```

FIELDNAM is the eight-character name of the field.

OUTLIN is the four-byte outlining specification. Valid values are:

```
O - OVERLINE
U - UNDERLINE
L - LEFT LINE
R - RIGHT LINE
```

The following return codes are received in SPARTNCD:

```
 0 -   The attributes were set correctly.
 4 -   Field not found.
 8 -   Invalid outlining parm.
16 -   Input Transaction Rule not expanded.
```

EXTENDED DATA BASE CALL SETUP ROUTINES

There are five extended call functions. The Extended Data Base Call Setup routines give the SPR the ability to perform complicated data base calls. Through these routines, the user can supply data base segment search arguments or control segment qualification. The entry points and their functions are:

```
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 ITR.

SETSSA - Set up user-segment search arguments.

SETUNQ - Unqualify some number of data base levels.

RSETSEGH - Resets the effect of any previous setup calls.

For all calls, with the exception of SETSSA, the supplied target segment ID must have a Segment Handler Rule. For SETSSA, if the target segment ID is defined in the ITR, I/O is allowed to/from the segment area as long as path call command codes are not present in the SSAs. These functions are "setup" routines. The actual data base I/O is not invoked until a SEGHNDLR call is issued against the same target segment ID. SETCC, SETPATH, and SETUNQ can be issued against the same target segment ID and will have a cumulative effect. The settings remain in effect until the TID (target segment ID) changes, a RESETSEGH call is issued, or until a subsequent SETCC, SETPATH, or SETUNQ is requested.

SETCC (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 call. (Note: This allows three command codes per data base level.)

The format of the SETCC call and its parameters are:

```
CALL SETCC (TID,ARRAY1[,ARRAY2]);        (in PL/I)

CALL SETCC USING TID, ARRAY1[,ARRAY2].   (in COBOL)
```

TID = the target segment ID. A Segment Handler Rule (SHR) must exist for this segment.

ARRAY1 = 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 = same as ARRAY1 except that this array is for the second command code. This parameter is optional.

SETPATH (Set Path Calls)

SETPATH is used to set up a path call operation against a target segment that is defined in the ITR. 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:

```
CALL SETPATH   (TID[,ID1,...IDn]);     (in PL/I)

CALL SETPATH USING TID[,ID1,...IDn].   (in COBOL)
```

TID = the target segment ID.

ID1-IDn = IDs of other segments involved in the path call.

A return code of 4 indicates that the SPA segment area cannot be used for the data base I/O.

The defined paths are controlled by the DBPATH and TSEG parameters on the Rules Generator GENERATE statement. The following examples show what the defined paths will be for the segments shown in Figure 6-3.

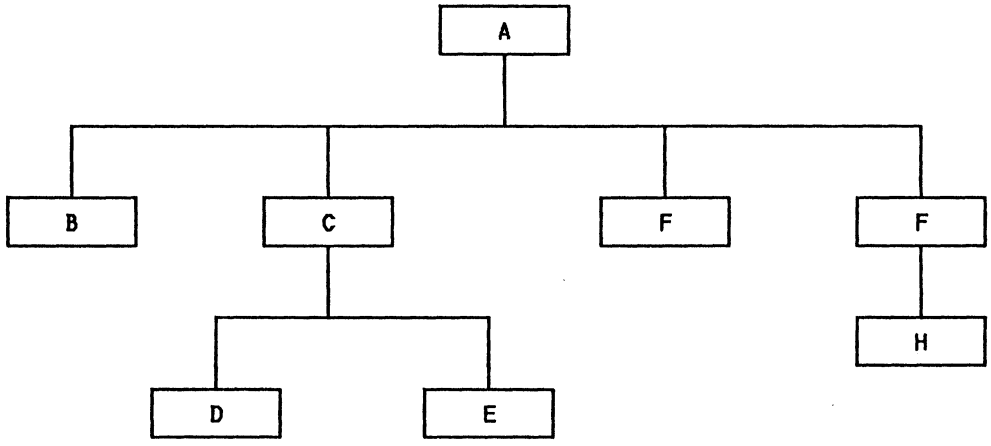


Figure 6-3. Examples of Defined Paths

Example 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

Example 2: DBPATH=(E,B,D)

The paths, as defined to IMSADF II, will be:

1-A,C,E 2-B 3-D

Example 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 ITR (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 only allowed 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 must be used if a delete is done immediately following a path retrieval that holds segments. The following steps should be taken to assure that the correct segment is deleted:

1. Issue a SETPATH call that only specifies the ID of the segment to be deleted.

2. Issue a SEGHNDLR call specifying the same ID and the DLET function.

SETSSA (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 ITR. When a match cannot be made on the last SSA segment name, the user is notified via a return code and TID field is blanked out. Also, I/O cannot be done in the segment area.

The call format is:

```
CALL SETSSA(TID,SSA1[,SSA2,...,SSAn]);              (in PL/I)

CALL 'SETSSA' USING TID, SSA1[, SSA2, ..., SSAn]).  (in COBOL)
```

TID = target segment ID. Will be set to blanks on return if not found in the ITR.

SSA1...SSAn = 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 second form of SEGHNDLR call, i.e., (ID,FUNC,PLD#[,UAREA]).

The return codes for SETSSA are:

0 - Successful completion.
4 - TID blanked out, segment name not found in ITR.
8 - Segment name found in ITR. Target segment ID was changed to match the segment ID found in the ITR. This occurs only if the TID and SSA segment name do not correspond.

SETUNQ (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:

```
CALL SETUNQ(TID,ARRAY);             (in PL/I)

CALL 'SETUNQ' USING TID, ARRAY.     (in COBOL)
```

where:

TID = target segment ID. A Segment Handler Rule (SHR) must exist for this ID.

ARRAY = 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).

The return codes for SETUNQ are:

0 - Successful Completion.
4 - No SSAs were set unqualified.

RSETSEGH (Reset Call (RSETSEGH)

RSETSEGH clears all previous data base setup calls. This is also done automatically in the following cases:

1 - a new TID in either a set call or a SEGHNDLR call.

2 - a SETPATH call after a SETCC call that contained path call command codes.

3 - a SETSSA call.

4 - SETCC, SETPATH, or SETUNQ call after a SETSSA call.

5 - on a subsequent SETCC, SETPATH, or SETUNQ call.

The format is:

```
CALL RSETSEGH
```

No parameters are required.

AUTOMATIC DATA BASE UPDATING ROUTINES

Two calls are provided that allow the user to utilize the Data Base Update routine for updating segments contained in the SPA segment area. SEGUPDTE is used to invoke this routine. SETFLAG is used, when necessary, to set indicators that control the updating process. With the exception of deleting segments, the SETFLAG routine normally won't be used. This is discussed later in this section.

SEGUPDTE

A single call to SEGUPDTE will cause all segments in the IMSADF II work area to be scanned to see if they should be inserted, replaced or deleted.

To invoke the Data Base Update routine, the call format is:

```
CALL SEGUPDTE
```

No parameters are required.

Return codes are:

-4 - Segment Handler Rule not found in Batch Driver Rule.

0 - Successful completion.

4 - Non-blank status code returned.

12 - Data compare failure in Conversational processing.

16 - No segments found to update.

A deletion requires that a previous call be issued to flag this segment for deletion. This function also invokes the data comparison logic for conversational transactions for which the DATACOMP=YES Rules Generator parameter has been specified.

Note: If an error is detected after some successful updates have been performed, a message is sent via the express PCB and a DL/I ROLL

function is issued to back out the successful updates and terminate the transaction.

SETFLAG

The indicators are set by the following call:

```
CALL SETFLAG(ID,FLAG,SETTING);              (in PL/I)

CALL 'SETFLAG' USING ID, FLAG, SETTING.     (in COBOL)
```

ID = two-character segment ID to have indicator set.

FLAG = one-character that defines which indicator is to be set. 'D' = delete flag. 'R' = retrieved flag. 'C' = changed flag. 'D' is the default.

SETTING = one character that defines how the indicator should be set. '0' = off. '1' = on. '1' is the default with 'D'. '0' is the default with 'R' and 'C'.

Return codes are:

0 - Successful completion.

4 - A dependent of the target segment has been changed and 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, the "segment changed" flag is used to determine how the data base segments are to be updated. This changed flag is set any time a field is changed by either a MAPPER or an AUDITOR request. The DL/I functions performed, based on the corresponding flag settings, are:

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 Rules Generator parameter DLET of the GENERATE statement, is used to define what segments are eligible for deletion. To prevent a segment from being either inserted or replaced (as determined by the setting of the retrieve flag) the changed flag must be set off. 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 parameter of the Rules Generator GENERATE statement. In order for the original data to be saved for the comparison, the segment must be retrieved into the SPA. The following summarizes when these indicators are set on or off:

changed flag ON

- auditing (value moved to a field)
- input from segment display screen or batch input
- mapping if field is changed
- SETFLAG routine

OFF

- successful DL/I call via SEGHNDLR or SEGUPDTE
- SETFLAG routine

delete flag ON

- SETFLAG routine
- delete call from the Auditor

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 SEGHNDLR or SEGUPDTE
- SETFLAG routine

OFF

- successful DL/I delete call via SEGHNDLR or SEGUPDTE
- key changed when mapping into segment area
- SETFLAG routine

SPECIAL PROCESSING ROUTINE EXAMPLE

Figure 6-4 depicts the logic required in a SPR to process all twin occurrences of a segment type, accumulate fields and display the totals. The logic follows:

The parent segment of the twin occurrences is retrieved by the appropriate transaction driver and the SPR is called. The first twin segment is retrieved by the SPR through a call to the Segment Handler. The SPR moves the appropriate fields from the SPA to the work area in the SPR by calling the Data Mapper. The fields are added to counters in the SPR and the next segment is retrieved by calling the Segment Handler again. This process is iterated until no more segments are found under the parent segment. The SPR accumulation counters then are moved to the SPA through a call to the Data Mapper, after which the SPR returns to the transaction driver with the specification that the screen is to be displayed.

The following areas are shown in this example:

- Data base segment retrieval
- Mapping data fields between the SPA and SPR work area
- Pseudo segments

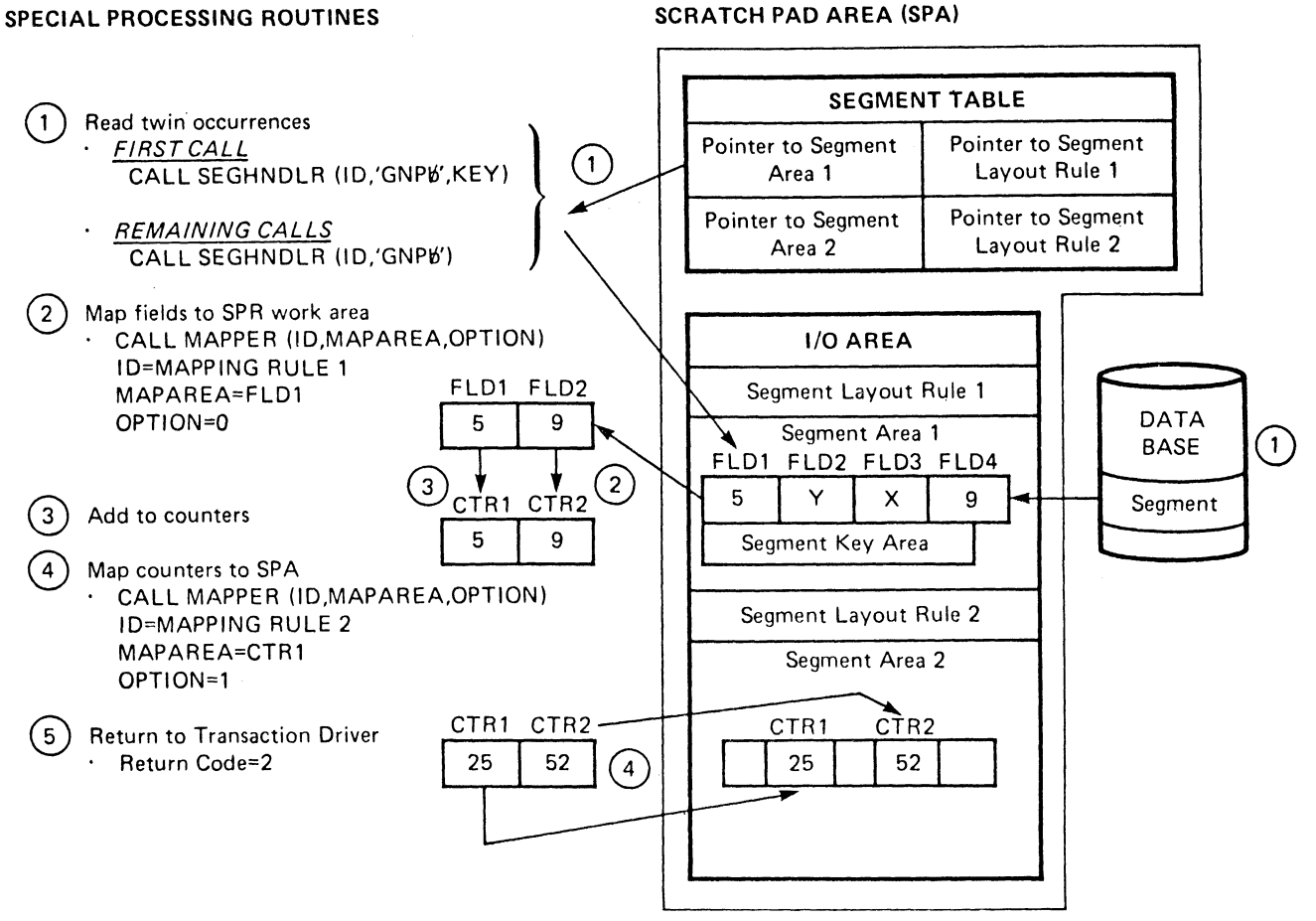


Figure 6-4. Special Processing Routine Example

Before the SPR gains control, the transaction driver performs the following tasks:

- Loads the Input Transaction Rule.
- Retrieves the parent segment for the twin occurrences that will be processed. Note: This segment is not shown in the SPA layout.
- Reserves space segment in the SPA for the twin segments to be read from the data base (Segment Area 1).
- Loads the Segment Layout Rule that describes the data base segment.
- Reserves space in the SPA for the pseudo segment (Segment Area 2).
- Loads the Segment Layout Rule that describes the pseudo segment.

The logic of the SPR is as follows:

Step 1

- FIRST CALL. Retrieve the first occurrence of the twin segments. Since this segment was not retrieved previously, the key parameter is required.
- REMAINING CALLS. Subsequent retrievals can be made without the key parameter, since the key is now contained in the Segment Key Area in the SPA.

Step 2

- Move the two required fields from the segment to the SPR work area. The ID parameter points to the appropriate mapping rule name in the Mapping Rule Table. MAPAREA points to a contiguous area in the SPR.

Step 3

- Add the two data base fields to the appropriate SPR counters. Return to Step 1 (REMAINING CALLS) to read the next segment occurrence.

Step 4

- A return code of 4 passed back from the Segment Handler in Step 1 (REMAINING CALLS) designates that all occurrences have been read. Move the two accumulation counters to the SPA. The ID parameter points to another mapping rule name in the Mapping Rule Table.

Step 5

- Return to the Transaction Driver with a return code of 2. This will direct the Transaction Driver to display the two counters as specified in the Input Transaction Rule.

PROCESSING WITH THE NONCONVERSATIONAL TRANSACTION DRIVER

Either Standard or Special Transactions Processing can be accommodated by the Nonconversational Transaction Driver. Standard Special Processing requires a user-written routine to handle unique or complex logic. The common functions available to the Driver are:

- auditing
- segment handling
- automatic message sending
- error message printing or display
- secondary transaction generation

The common functions and the language interface available to the Special Processing Routine and implemented with the Nonconversational Transaction Driver are described in the Special Processing section of this chapter.

Note: When a formatted screen is processed by the nonconversational transaction driver and an audit or rule error is encountered, an error screen is generated.

INPUT AND OUTPUT

Input to the nonconversational transaction driver is a single segment transaction message. Message input can be a maximum of 1600 bytes. The following is the layout of the message:

```
LLZ1Z2  ssssTmcc  mxx  FIELD1 FIELD2...FIELDn

   1        2      3          4

1  =     four-byte length field

         LL  =   message length

       Z1Z2  =   reserved bytes

2  =     transaction name, ssssTmcc, where

       ssss  =   major application system identification

          m  =   constant '0'

         cc  =   cluster code to identify the PSB for Standard
                 Processing or Special Processing ID for Special
                 Processing

3  =     mxx where

          m  =   the processing mode

         xx  =   Segment ID or Special Processing ID

4  =     input data fields whose relative position and length are
         defined by an Input Transaction Rule.  Note that a field
         will not be mapped into a data base segment or a pseudo
         segment unless it contains characters other than blanks
         defined as alpha or alphanumeric.  However, if blanks are
         desired in a data base field, that field may be blanked in
         one of the following ways:

         1.  if the input field contains a '#' in the first position
             followed by blanks to fill out the field; or

         2.  if the entire input field contains underscore ('_')
             characters.
```

The message format described above can be generated through either data entry or a secondary transaction generated by the Conversational Driver. If IMS/VS Message Format Service (MFS) is used to build the input message, option 1 is specified on the MSG statement so that MFS returns to the Nonconversational Transaction Driver all fields defined in the MID with data, data and fill characters, or fill characters. Output from the Nonconversational Driver consists of the following:

- Data base updates for eligible segments as specified in the Input Transaction Rule.
- Error messages routed to the 3284/3286 printer identified in the alternate IOPCB or to the display terminal if this is a response transaction.
- Response message using the Output Format Rule, if such response is indicated in the Input Transaction Rule.

- Secondary Transaction or alternate output message generation as indicated in the Input Transaction Rule.

RULES

The rules available for implementing a nonconversational application are the following:

- Input Transaction Rule
- Segment Layout Rule
- Segment Handler Rule
- Output Format Rule
- Audit Data Base
- Message Data Base (error message generation, automatic message sending, secondary transaction routing)

An Input Transaction Rule is required for each transaction ID or Special Processing ID to be processed by the Nonconversational Driver. This rule describes the layout of the fields in the transaction message, what data base of pseudo segments are needed to process the transaction message, what outputs are requested as secondary transactions or MFS outputs, and any special auditing requirements. The Segment Layout Rules describe the layout of data base segments and pseudo segments for Standard or Special Processing and also the layout of the mapping segments for Special Processing. The Segment Handler Rules describe the SSAs for retrieving or updating data base segments. The Output Format Rule describes the layout of the response message or a secondary transaction message that is routed to a terminal or a program as indicated by the Message Data Base. Auditing of input data, error message generation, and automatic message sending are achieved through entries in the Audit and Message Data Bases and through specifications in the Input Transaction and Segment Layout Rules.

PROCESSING WITH THE BATCH TRANSACTION DRIVER

The Batch Transaction Driver accommodates both Standard and Special Transaction Processing in a batch environment. Standard Processing is handled entirely through rule specifications, while Special Processing requires a user-written routine to handle unique or complex logic.

The Special Processing Routine is called by the Special Processing Interface Routine according to the Input Transaction and Batch Driver Rule specifications. Refer to the "Linkage Conventions" described under the "Special Processing Routine" section above. Note that the SPA is a data area in the Batch Transaction Driver. For compatibility, it has the same layout as the SPA used in online conversational processing. Special Processing Routine IDs are listed in the Batch Driver Rule and in the appropriate Input Transaction Rules for batch execution.

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 the user specifies EOF=Y in the appropriate Input Transaction Rule. For this maintenance call, the batch transaction driver calls the Special Processing Routine through the SPIR 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.

SIGN-ON PROCESSING IN BATCH MODE

A user may request that sign-on processing be required when processing in Batch Mode. This is accomplished by specifying SIGNON=YES on the GENERATE statement for the Batch Driver Rule (OPTIONS=BDLE) during the Rules Generation process. Then, each time that Batch Driver Rule is executed, a SIGNON transaction must be included in the input stream. The format of this transaction has been previously defined. It must appear in the input stream before any data base transactions. Additional SIGNON transactions may appear throughout the input, but they must never appear within a data base transaction.

The validation of the sign-on data entered by the user proceeds as in Conversational processing, with the following exceptions:

- The four-character application system identification in the PG segment of the Sign-on Profile Data Base must match the first four characters of the name of the Batch Driver Rule. If it does not, the sign-on is invalid.

- The Secondary Option Menu screen is not used. However, the transaction ID on each data base transaction (bytes 7-8 of the input) must match an entry in the PR segment(s).

- No messages are collected for a Project/Group via Project/Group message sending (exclusively a Conversational Processing option on the Primary Option menu). However, messages may be sent via Special Processing on the Auditor.

If any errors are encountered during sign-on processing, an appropriate error message is printed and all data base transactions are flushed until a valid SIGNON transaction is reached.

OPTIONAL LOCKWORD EXIT PROCESSING IN BATCH MODE

The user may include a lockword in the SIGNON transaction record. Verification is achieved through a user-written program in Assembler, PL/I, or COBOL that is called by the Batch Sign-on Module (MFC1TS9). The parameters that are passed to the lockword exit are the same in Batch as in Conversational Processing. The discussion of lockword processing in Conversational Processing applies to Batch Processing with the following exceptions:

- The skeleton batch PSB (BUILDT09 or BUILDT9B) supplied in the PSB source library contains a data base PCB which is used in lockword processing. The user must modify that PCB in the Batch PSB for each application which requires a lockword exit. Perform the PSBGEN.

For PL/I Lockword exits:

- To incorporate the user lockword program with the batch transaction 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 are the identifying characters for the Batch Driver Rule and ssss is the major system ID.

If your sign-on exit was written in 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 MFC1E01.

SIGN-OFF EXIT IN BATCH MODE (OPTIONAL)

A user exit may be invoked at sign-off time in Batch processing if a user has signed on. "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 exception that the parameter ADDR(NEWPG) is set to 0 since it does not apply to Batch. The information that is available in the SPA is the same that is available when Option 'C' is entered in Conversational mode.

To include a sign-off exit with the batch transaction 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(sign-off exit)
CHANGE      MFC1E99(sign-off exit)
INCLUDE     OBJLIB(MFC1E99P)
CHANGE      MFC1E99C(MFC1E99P)
INCLUDE     SYSLMOD(ssssBDxx)
ENTRY       MFC1T09
NAME        ssssBDxx(R)
```

where xx are the identifying characters for the Batch Driver Rule and ssss is the major system ID.

If your sign-off exit was written in 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.

CHECKPOINT AND RESTART IN SPECIAL PROCESSING ROUTINES

The batch SPR checkpoint restart support is designed for use in long-running special processing transactions. For standard processing and shorter special processing transactions, refer to Chapter 2 of the <u>IMS Application Development Facility II Version 2 Release 2 User Reference</u>.

A batch SPR can be designed to do checkpoint and restart processing with IMSADF II support. The SPR must use CHKPT=YES but not FREQ= on the BDLE GENERATE statement and follow a programming discipline of calling CHECKPT each time that a checkpoint is to be taken. The program must also check for a value of -2 in SPAFIRST and issue a RESTART call at that time to get the restart data from IMSADF II.

CHECKPT CALL

All calls to the CHECKPT routine from a particular SPR will have the same set of parameters. The format of the call is:

```
CALL CHECKPT(L1,A1,L2,A2,L3,A3, ...); (in PL/I)

CALL 'CHECKPT' USING L1,A1,L2,A2,L3,A3, ... (in COBOL)
```

Where

L1,L2,L3, ... are fullword binary fields containing lengths of the user areas to be included in the checkpoint.

A1,A2,A3, ... are the respective names of user areas to be included in the checkpoint.

RESTART CALL

The RESTART call in the SPR passes parameters corresponding to those on the CHECKPT calls. When the SPR is called, with a SPAFIRST value of -2, it calls RESTART to pick up the checkpointed values. The SPR saves these values until the appropriate call is made after which it can actually resume processing.

CHECKPOINT

When the SPR does a call to CHECKPT, IMSADF II will do a checkpoint call to IMS/VS. User and IMSADF II data areas will be recorded in the checkpoint for possible restart processing.

RESTART

IMSADF II will do restart processing for its areas when the batch driver begins execution. It will locate the transaction that was executing when the last checkpoint was taken (the SPR) and invoke that transaction. When the SPR makes the initial call to RESTART, IMSADF II will restore the SPR data areas to their values according to the checkpoint information. The SPR must then handle its own logic and recovery processing. Notice that the design of the SPR must account for any IMSADF II key selection done before calling the SPR and BYPASS=YES coding on the generate statement. For this reason, the SPAFIRST value should be part of the data saved. Changes in data base positioning could impact the execution of the SPR.

For SPR control of checkpoint and restart, the RSTRTIN DD statement must be used. Information required for the restart is contained in the restart record in the file. The record layout is the same as noted in Chapter 2 of the <u>IMS Application Development Facility II Version 2 Release 2 User Reference</u> with the addition of a user area field located in columns 22 through 27. The number in this field tells IMSADF II how many bytes of user data were saved in the checkpoint and also tells IMSADF II that SPR restart is to be done.

```
---- ----1---- ----2---- ----3
RSTRT A8N00213       003000
```

<u>DL/I EXIT PROCESSING</u>

The DL/I Exit capability provides an exit point before and after each data base DL/I call issued during the processing of a user's transaction. This exit may be utilized to perform such functions as:

building a history file of user data base activity,

modifying a PCB address for mirror image (split) data bases,

modifying Segment Search Arguments (SSA),

manipulating segment I/O areas.

The exit can be invoked for both standard and special processing transactions. It applies to conversational, nonconversational, and batch transaction drivers. The exit is link-edited with the appropriate mini-driver by the Rules Generator. Invocation of the DL/I Exit is requested by specifying DLIEXIT=YES on the Rules Generator GENERATE statement that defines a transaction.

The DL/I Exit can be written in the following programming languages:

- COBOL (with NODYNAM,NOENDJOB options)
- PL/I (must be compiled as a non-main program)
- ASSEMBLER

INVOCATION

A DL/I Exit is invoked during a user's transaction ID TRXID processing. The DL/I Exit is link-edited with the appropriate mini-driver by the Rules Generator and started according to the Input Transaction Rule specification. Since the DL/I Exit is link-edited with the mini-driver, there is one DL/I Exit for an application system mini-driver and cluster code (SOMTX operand value). If the DL/I Exit's logic requires sensitivity to transaction mode (1 to 6) or ID, the SPA will be available to determine the currently processed transaction mode and ID. (Refer to the SPATRX variable in the SPACOBOL, SPAPLI or SPAASM layouts in library IMSADF.ADFMAC, distributed with IMSADF II).

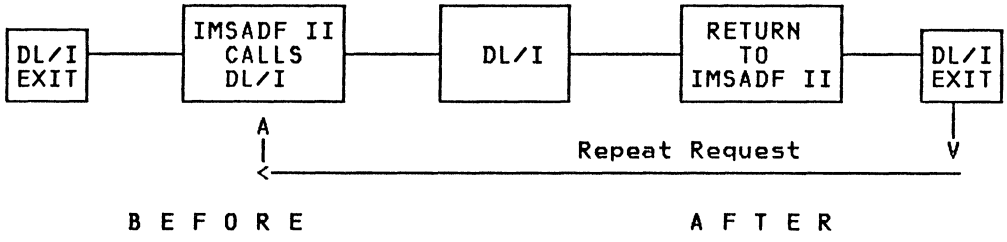
The DL/I Exit is invoked before and after the following DL/I data base calls:

- Data base retrievals and updates, whether performed by the IMSADF II transaction driver, requested by audit rules or issued through SEGHNDLR calls from audit exits or special processing routines.
- Secondary Key selection
- Audit Data Base calls if the static audit rule is not used
- Message Data Base calls for errors, secondary transaction sending, informational messages to a user ID
- Sign-on Profile Data Base calls for informational messages to a project/group and for security checks on a TRXID change

DL/I data base calls, not included in the Exit processing, are:

- IMSADF II Work Data Base for SPA processing
- Sign-on Profile Data Base for conversational sign-on processing

The DL/I Exit is invoked before and after each DL/I call issued during the transaction processing. Additionally, the DL/I Exit can request IMSADF II to reissue the DL/I call following the AFTER call. Following is a graphic representation of the DL/I Exit invocation.



For Fast Path Data Bases (Main Storage and Data Entry), the SSALIST represents the full function DL/I SSA's, i.e. command codes and multiple SSA's. For those DL/I calls resulting in a reformatted SSALIST, the DL/I Exit is invoked once before the Fast Path DL/I call or calls and once after the Fast Path DL/I call or calls. An example of the DL/I Exit invocation and a GU DL/I call for a second level segment in a Data Entry Data Base (DEDB) follows:

```
DL/I Exit (BEFORE) SSALIST=full function DL/I, IOAREA=Dependent's area
                   |
                   V
DL/I call GU   Root      Fast Path SSALIST
DL/I call GNP  Dependent Fast Path SSALIST
                   |
                   V
DL/I Exit (AFTER)  SSALIST=full function DL/I, IOAREA=Dependent's area
```

For DL/I calls resulting in a 'GG' status code, the following DL/I Exit invocation pattern is followed:

```
DL/I Exit (BEFORE)
        |
        V
DL/I Call (GG Status)
DL/I Call (GG Status) RETRY 1
DL/I Call (GG Status) RETRY 2
        |
        V
DL/I Exit (AFTER )
```

Parameters

The following parameter addresses are passed to the DL/I Exit routine:

USERINT	3F binary	USERINT is a table with 3 values. TIMECALL F bin 0=before, 1=after call PCB# F bin Relative PCB 1 to 120 CALLSTAT F bin 0=continue, 1=repeat
DLICNT	F binary	number of parameters for DL/I call DLICNT= 3 + number of SSAs
DLIFUNC	4 bytes	DL/I function, i.e. GU,GHU,REPL
PCBMASK	36 bytes + max concat key	application program mask length = 36 + maximum concatenated key length
SEGIOA	variable	segment I/O area
SSALIST 1 through SSALIST15	variable	Segment search arguments. Fifteen addresses are passed. DLICNT indicates how many are valid
SPA	800 bytes + COMMLEN	SPA layouts (SPAASM,SPAPLI,SPACOBOL) are included in MACLIB
IOPCB	40 bytes	Terminal PCB Mask
ALTIOPCB	40 bytes	Alternate Terminal PCB
EXPIOPCB	40 bytes	Express Terminal PCB
PCBCNT	F binary	number of user PCBs that follow (1 to 120) PCBCNT is not included in the count
USERPCB (1 to 120)	36 char + max concat key	IMS/VS application Program Control Block (PCB) mask

- USERINT -is a three-word table defining a before/after call, the PCB to be used, and whether the ASMTDLI call should be repeated.
  - TIMECALL - indicates whether this invocation of the DL/I Exit is before or after the DL/I call. 0=before, 1=after.

- PCB# - indicates the relative number (1 to 120) of the user PCB to be used or that was used in the DL/I call. For the IMSADF II data bases, the value will be 0. To change the PCB for this DL/I call, the user Exit changes the value in PCB#.

- CALLSTAT - indicates to the IMSADF II Common Segment Handler routine, whether the DL/I call should be repeated. This can be used when a mirror image (split) data base application encounters a GB status code and wants to continue retrievals in the next data base. The Exit would set CALLSTAT to 1 and the PCB# to the next physical data base. This variable only applies to the 'after' call. At entry to the EXIT, CALLSTAT is set to 0. On return to IMSADF II, 0 means continue and 1 means repeat the DL/I call.

- DLICNT - contains the number of parameters for this DL/I call. DLICNT includes the DLIFUNC, PCBADDR, SEGIOA, and (1 to 15) SSAs. The count in DLICNT does not include itself.

- DLIFUNC - contains the four-byte DL/I data base function, GU, GHU, etc.

- PCBMASK - contains the IMS/VS PCB mask for this DL/I call.

- SEGIOA - contains the segment I/O area for this DL/I call. It contains the data for one or more segments. The existence of a path call command code (D) in the SSAs will indicate additional segment I/O areas may be present.

- SSALIST (1 through 15) - contains the SSA for this DL/I call. Fifteen SSAs are always passed to the Exit. DLICNT indicates the number of parameters for the DL/I call. Those SSAs not used in the DL/I call contain a blank in position 1.

  Following are the formats of the SSAs used by IMSADF II:

  - Qualified DL/I Call

    xxxxxxxx*---(ffffffff =value) xxxxxxxx is the eight-byte segment name in the IMS/VSDBD (Data Base Description) control block and in the NAME operand of the IMSADF II Rules Generator SEGMENT statement. ffffffff is the eight-byte sequence (or search) field name in the DBD and the NAME Operand of the KEY field in the Rules Generator FIELD statement. For path calls, xxxxxxxx*D--(ffffffff =value) will be the SSA structure. For update calls, where a segment in the path is not replaced, the command code N is used in place of the D. The last two dashes -- contain any user command codes.

  - Unqualified Call

    xxxxxxxx*--- where xxxxxxxx is the eight-byte segment name in the DBD and the NAME operand of the Rules Generator SEGMENT statement. The left parenthesis '(' has been replaced with a blank.

- SPA - is the Scratch Pad Area for conversational transactions and the work area for nonconversational and batch transactions. The EXIT has access to the SPA variables used by IMSADF II and the communication area.

- IOPCB - contains the IMS/VS PCB mask for the terminal PCB.

- ALTIOPCB - contains the IMS/VS PCB mask for the modifiable alternate PCB. This parameter does not apply to Batch Driver transactions, unless the Batch Driver is running as a Batch Message Processing program (BMP) and BUILDT9B was used to generate the Program Specification Block (PSB).

- EXPIOPCB - contains the IMS/VS PCB mask for the EXPRESS PCB. This parameter does not apply to batch driver transactions, unless the Batch Driver is running as a Batch Message Processing program, BMP, and BUILDT9B was used to generate the PSB.

- PCBCNT - contains the number of user PCBs that follow. PCBCNT is not included in the count.

- USERPCB (1 to 120) - contains the user data base PCBs passed from IMS/VS. The number of USERPCBs is variable (1 through 120). The remainder of the PCBs, through 120, will contain eight blanks, i.e., the data base name will be eight blanks.

PARAMETER MODIFICATIONS

The following parameters can be modified by the DL/I Exit:

- PCB# - On entry to the DL/I Exit, PCB# is set to the relative PCB number (1-120) of the user data base, to be used, or that was used, in the DL/I call. If PCB# is modified by the EXIT on the BEFORE call, the Common Segment Handler module uses the new PCB. If PCB# is modified on the AFTER call, CALLSTAT must be set to 1 to indicate that the DL/I call should be repeated with a new PCB#.

- CALLSTAT - On entry to the DL/I Exit, CALLSTAT is set to 0. If the EXIT, on the AFTER call, determines that another DL/I call should be issued by IMSADF II, CALLSTAT should be set to 1. Additionally, if a different PCB# is required, PCB# should be modified concurrently. CALLSTAT does not apply to the BEFORE call.

- SEGIOA - SEGIOA contains the segment data to be passed to DL/I by IMSADF II or the segment data retrieved by DL/I before processing by IMSADF II. The DL/I Exit can modify this data, but those modifications must be consistent with the Rules Generator definitions for these segments with respect to segment length, field length, field data type, and field order.

- SSALIST1 to SSALIST15 - Each SSALIST contains the SSA for a segment in the hierarchical path. SSALIST can be modified but should be consistent with its DBD definition to IMS/VS. Note that if the key values are modified in the SSALIST, the keys are not updated in the concatenated key areas maintained by IMSADF II. Modification of SSALIST applies only to the BEFORE call.

EXIT FUNCTIONS AVAILABLE

The following exit interface functions are available to the DL/I Exit:

- MAPPER (MAPCODE=0 map from SPA to user exit)

- COPYSEG (COPYCODE=0 copy from SPA to user exit)

- GETKEY

- GETKINFO

The DL/I Exit can issue native DL/I calls using the DL/I language interface, ASMTDLI, CBLTDLI, or PLITDLI to access another data base in the user PCB list. The Exit should not modify PCB positioning on the current call by issuing its own ASMTDLI, CBLTDLI, or PLITDLI for the current PCB.

Note that the functions, AUDITOR, SEGHNDLR, and SEGUPDTE, that could result in another DL/I call being issued by IMSADF II, can not be used within the DL/I Exit. Additionally, the keys in the segment I/O areas, and current keys keys maintained by IMSADF II should not be modified.

RULES GENERATOR OPERANDS

Invocation of a DL/I Exit is controlled at the transaction level. Specification of DLIEXIT=YES on the the GENERATE Statement for a transaction causes the Common Segment Handler module to invoke the DL/I Exit that has been link-edited with the mini-driver.

```
DLIEXIT=NO      (default)
        YES
```

The DLIEXIT operand can be coded on those GENERATE statements that have OPTIONS=INTR,CVALL,TPIT,TPALL,or BAIT.

To link-edit a DL/I Exit with the mini-driver, compile the DL/I Exit and place the object module or load module (depending on installation definition RGLIB) in a partitioned data set (PDS). Then run the Rules Generator to link-edit the DL/I Exit routine with a copy of the appropriate mini-driver (conversational, nonconversational, or batch). The OBJLIB DD or SYSLIB DD statement in the Rules Generator JCL must point to the PDS containing the DL/I Exit, since the Rules Generator builds a Linkage Editor control statement:

```
INCLUDE  OBJLIB(xxxxxxxx) where xxxxxxxx is the EXIT name
     OR

INCLUDE  SYSLIB(xxxxxxxx)
```

The following operands are used on a GENERATE statement with OPTIONS=STLE,SPLE,NCLE,TPLE, or BDLE to link-edit the DL/I Exit with the appropriate mini-driver.

```
DEXIT=xxxxxxxx where xxxxxxxx is the name of the DL/I
               Exit routine

DLANG=COBOL   (default)
      ASMSTD
      ASMINT
      PL/I
```

For conversational special processing transactions that are clustered (OPTIONS=SPLE, DYNLOAD=YES), the DL/I Exit is link-edited with the associated standard driver (OPTIONS=STLE).

DL/I EXIT EXAMPLES

Following is an example of a COBOL program accepting the DL/I exit parameters:

```
      IDENTIFICATION DIVISION.
       PROGRAM-ID.
          BANKDCOB.
       DATE-COMPILED.
       REMARKS.
          COBOL DLI USER EXIT.
      ENVIRONMENT DIVISION.
       CONFIGURATION SECTION.
        SOURCE-COMPUTER. IBM-370.
        OBJECT-COMPUTER. IBM-370.
      DATA DIVISION.
      LINKAGE SECTION.
     *   DL/I EXIT INTERFACE TABLE
      01  DEXIT.
          02 TIMECALL        PICTURE S9(9) COMPUTATIONAL.
          02 PCBNBR          PICTURE S9(9) COMPUTATIONAL.
          02 CALLSTAT        PICTURE S9(9) COMPUTATIONAL.
     *   DL/I PARMCOUNT
      77  DLICNT             PICTURE S9(9) COMPUTATIONAL.
     *   DL/I FUNCTION (GU,GHU,GN,ETC.)
      77  DLIFUNC            PICTURE X(4).
     *   DATA BASE PCB FOR THIS DLI CALL
     *   PCBMASK DEFINES THE DATA BASE PCB LAYOUT
      01  DBPCB
          COPY DBPCBMSK.
     *   SEGMENT IOAREA
      77  SEGIOA             PICTURE X(200).
     *   SSALIST  15 SSA'S - NBR(SSA'S) = DLICNT - 3
     *   SSAMASK =   02 SEGNAME PICTURE X(8).
     *               02 CCODES  PICTURE X(4).
     *               02 DELIM   PICTURE X.
     *               02 KEYNAME PICTURE X(8).
     *               02 RELOP   PICTURE X(2).
     *               02 VALUE   PICTURE X(255).
      01  SSA1.
          COPY SSAMASK.
      01  SSA2.
          COPY SSAMASK.
      01  SSA3.
          COPY SSAMASK.
      01  SSA4.
          COPY SSAMASK.
      01  SSA5.
          COPY SSAMASK.
      01  SSA6.
          COPY SSAMASK.
      01  SSA7.
          COPY SSAMASK.
      01  SSA8.
          COPY SSAMASK.
      01  SSA9.
          COPY SSAMASK.
      01  SSA10.
          COPY SSAMASK.
      01  SSA11.
          COPY SSAMASK.
      01  SSA12.
          COPY SSAMASK.
      01  SSA13.
          COPY SSAMASK.
      01  SSA14.
          COPY SSAMASK.
      01  SSA15.
          COPY SSAMASK.
```

```
*  SPA LAYOUT.
     COPY SPACOBOL.
*  IOPCB,ALTERNATE IOPCB, EXPRESS IOPCB
*  LTPCBMASK DEFINES THE LAYOUT OF THE TERMINAL PCB
 01  IOPCB.
     COPY LTPCBMSK.
 01  AIOPCB.
     COPY LTPCBMSK.
 01  EXPIOPCB.
     COPY LTPCBMSK.
*  COUNT OF USER PCBS THAT FOLLOW
 77  PCBCNT            PICTURE S9(9) COMPUTATIONAL.
*  USER PCBS (1 TO 120)
*  THE NUMBER OF USER PCBS DEFINED MUST ACCOUNT FOR THE
*  LARGEST PSB
 01  PCB1.
     COPY DBPCBMSK.
 01  PCB2.
     COPY DBPCBMSK.
 01  PCB3.
     COPY DBPCBMSK.
 01  PCB4.
     COPY DBPCBMSK.
 01  PCB5.
     COPY DBPCBMSK.
     EJECT
 PROCEDURE DIVISION USING DEXIT,DLICNT,DLIFUNC,DBPCB,SEGIOA,
     SSA1,SSA2,SSA3,SSA4,SSA5,SSA6,SSA7,SSA8,SSA9,SSA10,
     SSA11,SSA12,SSA13,SSA14,SSA15,SPADSECT,IOPCB,AIOPCB,
     EXPIOPCB,PCBCNT,PCB1,PCB2,PCB3,PCB4,PCB5.
*
*  program logic
*
     GOBACK.                          NOTE RETURN TO CALLER.
```

Following is an example of a PL/I program accepting the DL/I exit parameters:

```
BANKDPL:
   PROC(DEXITP,DCNTP,DFUNCP,DPCBP,DIOAP,DSSA1P,DSSA2P,DSSA3P,DSSA4P,
        DSSA5P,DSSA6P,DSSA7P,DSSA8P,DSSA9P,DSSA10P,DSSA11P,DSSA12P,
        DSSA13P,DSSA14P,DSSA15P,SPAP,IPCBP,APCBP,EXPCBP,PCBCNTP,
        PCB1P,PCB2P,PCB3P,PCB4P,PCB5P);
DCL
   PLIXOPT CHAR(50) VAR            /* SET UP EXECUTION TIME OPTIONS*/
      INIT('NOCOUNT,NOFLOW,NOREPORT,NOSTAE,NOSPIE')
      STATIC EXTERNAL;
DCL (DEXITP,DCNTP,DFUNCP,DPCBP,DIOAP,DSSA1P,DSSA2P,DSSA3P,DSSA4P,
     DSSA5P,DSSA6P,DSSA7P,DSSA8P,DSSA9P,DSSA10P,DSSA11P,DSSA12P,
     DSSA13P,DSSA14P,DSSA15P,SPAP,IPCBP,APCBP,EXPCBP,PCBCNTP,
     PCB1P,PCB2P,PCB3P,PCB4P,PCB5P) FIXED BIN(31);
  /*    DECLARES FOR POINTERS FOR BASING THE PARMLIST              */
DCL (DEXITA,DCNTA,DFUNCA,DB_PCBPTR,DIOAA,SSAPTR,
     LT_PCBPTR,PCBCNTA) POINTER;     /* SPAPTR DEFINED IN SPAPLI   */
  /*    DECLARES FOR PARAMETERS FROM ADF SEGMENT HANDLER           */
DCL 01 DEXIT BASED(DEXITA),
       02 TIMECALL FIXED BINARY(31), /* 0=BEFORE 1=AFTER           */
       02 PCB#     FIXED BINARY(31), /* 0 TO 120                   */
       02 CALLSTAT FIXED BINARY(31); /* 0=CONTINUE,1=REPEAT DLICALL*/
DCL DLICNT FIXED BINARY(31) BASED(DCNTA);/* COUNT OF DLI PARMLIST  */
DCL DLIFUNC CHAR(4) BASED(DFUNCA);   /* DLI FUNCTION               */
DCL 01 PCBNAME  BASED(DB_PCBPTR),
       02 DB_PCBDBD CHAR(8),         /* DBDNAME                    */
       02 DB_PCBLEV CHAR(2),         /* LEVEL OF RETURNED SEGMENT  */
       02 DB_PCBSTC CHAR(2),         /* DLI STATUS CODE            */
       02 DB_PCBPRO CHAR(4),         /* PROCOPT OPTIONS            */
       02 DB_PCBRSV FIXED BINARY(31),/* RESERVED                   */
       02 DB_PCBFD  CHAR(8),         /* SEGMENT NAME               */
```

```
        02 DB_PCBLKA FIXED BINARY(31),/* LENGTH OF KEY FEEDBACK AREA*/
        02 DB_PCBNSS FIXED BINARY(31),/* NO OF SENSITIVE SEGMENTS   */
        02 DB_PCBKFA CHAR(255);       /* LARGEST KEY FEEDBACK AREA  */
 DCL SEGIOA CHAR(200) BASED(DIOAA);   /* SEGMENT IOAREA             */
 DCL 01 SSA   BASED(SSAPTR),          /* SEGMENT SEARCH ARGUMENT    */
        02 SEGNAME CHAR(8),           /* SEGMENT NAME               */
        02 CCODES  CHAR(4),           /* COMMAND CODES *---         */
        02 DELIM   CHAR(1),           /* ( OR BLANK                 */
        02 KEYNAME CHAR(8),           /* KEY FIELD NAME             */
        02 RELOP   CHAR(2),           /* RELATIONAL OPERATOR        */
        02 VALUE   CHAR(255);         /* MAXIMUM KEY VALUE          */
        INCLUDE SYSLIB(SPAPLI);
 DCL 01 LTNAME   BASED(LT_PCBPTR),
        02 LT_PCBDBD CHAR(8),         /* LOGICAL TERMINAL NAME      */
        02 LT_PCBRSV CHAR(2),         /* RESERVED                   */
        02 LT_PCBSTC CHAR(2),         /* DLI STATUS CODE            */
        02 LT_PCBDAT CHAR(4),         /* CURRENT DATE               */
        02 LT_PCBTIM CHAR(4),         /* TIME                       */
        02 LT_PCBSEQ FIXED BINARY(31),/* INPUT MSG SEQUENCE NBR     */
        02 LT_PCBMOD CHAR(8),         /* MODNAME                    */
        02 LT_PCBUSR CHAR(8);         /* USERID OR PSBNAME OF BMP   */
 DCL PCBCNT FIXED BINARY(31) BASED(PCBCNTA);/* COUNT OF USER PCBS  */
        PAGE;
  /*   SET UP BASED POINTERS                                       */
        DEXITA=ADDR(DEXITP);          /* DLI EXIT INTERFACE TABLE   */
        DCNTA=ADDR(DCNTP);            /* COUNT FOR DLI PARMLIST     */
        DFUNCA=ADDR(DFUNCP);          /* DLI FUNCTION FOR THIS CALL */
        DB_PCBPTR=ADDR(DPCBP);        /* DLI PCB ADDRESS            */
        DIOAA=ADDR(DIOAP);            /* SEGMENT IOAREA             */
        SSAPTR=ADDR(DSSA1P);          /* SSA                        */
        SPAPTR=ADDR(SPAP);            /* SPA ADDRESS                */
        LT_PCBPTR=ADDR(IPCBP);        /* LOGICAL TERMINAL PCB       */
        PCBCNTA=ADDR(PCBCNTP);        /* USER PCB COUNT             */
        SKIP(2);
  /*    program logic                                               */
        RETURN;
 END BANKDPL;
```

Following is an example of an ASSEMBLER program accepting the DL/I exit parameters:

```
BANKDASM CSECT
*****************************************************************
*
*   PARAMETERS -  DEXIT,DLICNT,DLIFUNC,DBPCB,SEGIOA,15(SSAS),
*                 SPA,IOPCB,AIOPCB,EXPIOPCB,PCBCNT,USERPCB(1 TO 120)
*
*****************************************************************
         USING 3ANKDASM,R15
R0       EQU   0
R1       EQU   1
R2       EQU   2
R3       EQU   3
R4       EQU   4
R5       EQU   5
R6       EQU   6
R7       EQU   7
R8       EQU   8
R9       EQU   9
R10      EQU   10
R11      EQU   11
R12      EQU   12
R13      EQU   13
R14      EQU   14
R15      EQU   15
```

```
         SPACE
         STM   R14,R12,12(R13)      SAVE CONTENTS OF CALLERS REGS
         LA    R14,BANKSAVE
         ST    R14,8(R13)           POINT CALLERS SAVE TO BANKSAVE
         ST    R13,4(R14)           POINT BANKSAVE TO CALLERS SAVE
         LR    R13,R14              R13 POINTS TO BANKSAVE
         LR    R12,R15
         DROP  R15
         USING BANKDASM,R12         R12 IS NOW THE BASE REG
         SPACE
*   SET UP ADDRESSES FROM THE PARAMETER LIST
         MVC   PARMLIST(140),0(R1)  MOVE IN PARMLIST
         L     R6,DEXITA            ADDRESS OF INTERFACE TABLE
         USING DEXIT,R6
         L     R8,DLIPCBA           ADDRESS OF PCB FOR THIS DLI CALL
         USING PCBNAME,R8
*   set up addressing for other parameters as needed
*
*   program logic
*
RETURN   EQU   *
         SR    R15,R15
         L     R13,4(R13)           RESTORE CALLERS R13
         LM    R0,R12,20(R13)       RESTORE CALLERS REGS
         L     R14,12(R13)          RESTORE RETURN POINT
         BR    R14                  RETURN TO CALLER
         EJECT
*******************************************************************
*        ADDRESSES FOR PARAMETERS FROM V09
*******************************************************************
PARMLIST DS    0F
DEXITA   DS    A                    ADDRESS OF DEXIT INTERFACE TABLE
DLICNTA  DS    A                    ADDRESS OF DLICNT FOR DLI CALL
DLIFUNCA DS    A                    ADDRESS OF DLI FUNCTION
DLIPCBA  DS    A                    PCB ADDRESS FOR DLI CALL
DLIIOA   DS    A                    SEG IOAREA ADDRESS
DLISSAA  DS    15A                  15 SSA ADDRESSES
SPAPTR   DS    A                    ADDRESS OF SPA
IOPCBA   DS    A                    TERMINAL PCB ADDRESS
AIOPCBA  DS    A                    ALTERNATE TERMINAL PCB ADDRESS
EXIOPCBA DS    A                    EXPRESS TERMINAL PCB ADDRESS
PCBCNTA  DS    A                    ADDRESS OF USER PCBCNT
USERPCBA DS    10A                  USER PCB ADDRESSES (1 TO 120)
         LTORG
*******************************************************************
*        DSECT FOR DL/I EXIT INTERFACE TABLE
*******************************************************************
DEXIT    DSECT
TIMECALL DS    F                    0=BEFORE DLI CALL, 1=AFTER DLI CALL
PCB#     DS    F                    RELATIVE PCB 1 TO 120 FOR DLI CALL
CALLSTAT DS    F                    0=CONTINUE, 1=REPEAT DLI CALL
         SPACE
```

```
*****************************************************************
*        DSECTS FOR DL/I CALL
*****************************************************************
         SPACE
DLICNTD  DSECT
DLICNT   DS    F                    PARMCOUNT FOR DLI CALL
         SPACE
DLIFUNCD DSECT
DLIFUNC  DS    CL4                  DLI FUNCTION (GU, GHU, ETC)
         SPACE
SEGIOAD  DSECT
SEGIOA   DS    CL200                LENGTH OF SEGMENT IOAREA
         SPACE
SSAD     DSECT                      SSA LAYOUT
SEGNAME  DS    CL8                  SEGNAME
CCODES   DS    CL4                  COMMAND CODES *---
DELIM    DS    CL1                  ( OR BLANK
KEYNAME  DS    CL8                  KEY FIELD NAME
RELOP    DS    CL2                  RELATIONAL OPERATOR - BLANK
VALUE    DS    CL255                KEY VALUE )
         EJECT
         COPY  SPAASM               SPA LAYOUT IN ADF MACLIBF
         EJECT
*****************************************************************
*        DSECT FOR TERMINAL PCB
*****************************************************************
LTNAME   DSECT
LTPCBDBD DS    CL8                  LOGICAL TERMINAL NAME
LTPCBRSV DS    CL2                  RESERVED
LTPCBSTC DS    CL2                  STATUS CODE
LTPCBDAT DS    F                    CURRENT DATE
LTPCBTIM DS    F                    CURRENT TIME
LTPCBSEQ DS    F                    INPUT MSG SEQUENCE NUMBER
LTPCBMOD DS    CL8                  MOD NAME
LTPCBUSR DS    CL8                  USERID OR PSBNAME OF SENDING BMP
         SPACE
*****************************************************************
*        DSECT FOR DATA BASE PCB AND PCBCNT
*****************************************************************
PCBCNTD  DSECT
PCBCNT   DS    F                    CNT OF USER PCBS PASSED IN PARMLIST
         SPACE
PCBNAME  DSECT
DBPCBDBD DS    CL8                  DBDNAME
DBPCBLEV DS    CL2                  LEVEL FEEDBACK
DBPCBSTC DS    CL2                  STATUS CODE
DBPCBPRO DS    CL4                  PROC OPTIONS
DBPCBRSV DS    F                    RESERVED
DBPCBFD  DS    CL8                  SEGMENT NAME
DBPCBLKA DS    F                    LENGTH OF KEY FEEDBACK AREA
DBPCBNSS DS    F                    NO OF SENSITIVE SEGMENTS
DBPCBKFA DS    CL17                 KEY FEEDBACK AREA
         END
```

CONVERSATIONAL LOCKWORD EXIT SPECIFICATION PROCESSING

The IMSADF II Sign-on screen provides an eight-character alphanumeric field for entering a lockword. The lockword may be verified by a user-written program (ASSEMBLER, COBOL or non-MAIN PL/I procedure) that is CALLed by the Sign-on module (MFC1TS1). 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:

```
WORD1  = ADDR (INPUT)
WORD2  = ADDR (ERRMSG)
WORD3  = ADDR (SPA)
WORD4  = ADDR (DBPCB)
WORD5  = ADDR (SRSEG)
WORD6  = ADDR (IOPCB)
WORD7  = ADDR (ALTPCB)
WORD8  = ADDR (EXPPCB)
WORD9 =  ADDR (USER DB PCB LIST)
```

INPUT is an 80-byte input area from the Sign-on screen that specifies the user ID, Project/Group, lockword, and major application system identification.

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

The ERRMSG area is 44 bytes in length and is 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.ADFMAC as member name SPAASM, SPAPLI, or SPACOBOL.

DBPCB is the first or only data base PCB provided by the application developer. To add the PCB to the IMSADF II PSB, copy PSB, ????TOM, from library IMSADF.PSBSRC. Add the desired PCB just prior to the PSBGEN statement. Perform the PSBGEN for ????TOM. The lockword user exit routine will now have access to that PCB. Note ???? is the installed ADFID.

The SRSEG is the SR segment retrieved from the Sign-on Profile data base. Refer to Data Base Segment Layout above for the layout of the SR segment.

USER DB PCB LIST is the address of a <u>list</u> of user data base PCBs which you may use in the lockword exits, it includes the first one but is not limited to that one. To add those PCBs to the ????TOM PSB, follow instructions for DBPCB above. 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 other 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.

IMSADF II supplied a lockword exit for Sign-on called MFC1E01 and one for Sign-off (MFC1E99). Please note that the exits supplied by IMSADF II are 'dummy' exits, and that you are expected to replace them by yours when you code them. There should be no IBM maintenance to either of the modules which you are replacing. The IMS Application Development Facility II Version 2 Release 2 Installation Guide gives instructions for incorporating the exits into your IMSADF II system. Implementation of them takes two forms based on the language in which you wrote the exits.

When your exit is written in either COBOL or Assembler, it is incorporated into your IMSADF II system as an SMP Usermod, and maintenance on any modules which are part of MFC1TOM, the Sign-on processor, is accomplished without any manual intervention on your installation's part.

If you write your exit in COBOL or assembler, the name of your Sign-on exit is MFC1E01. Compile it and link-edit it with any subroutines you require in IMSADF.RULLIB.

If your Sign-on exit is written in PL/I, you are encouraged to give it the name of MFC1E01. When you use PL/I exits you must add a module for the interface to it. Since you are changing the link-edit structure for Sign-on, you must effect your change outside of the SMP environment. Your code will be a non-MAIN PL/I exit. Place the object text in a library of your choice, from which you will include it at link-edit time. The Linkage Editor control statements you use are supplied in the IMS Application Development Facility II Version 2 Release 2 Installation Guide.

There are several reasons for installing a lockword exit, and they 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 an 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 sign-on screen through the Rules Generator screen image capability.

The sign-on module, MFC1TS1, 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 or the project group. You must ensure that the proper language and SYSID are set. 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 then SPAULANG will be reset and the prefix used is that of ADFID. Selection of the eight common IMSADF II screens is done using the SPAMFSPF value set at Signon.

Note: 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.

Alternative Sign-on Display Format

Another way you may use the Sign on exit is to control screen flow after the processing for the current application is complete. A field called SPAMODSI contains the four-character name of the sign-on screen (MFS MOD name or BMS MAPSET name). If you have a non-IMSADF II generalized sign-on screen that you wish to present which differs from the original sign-on screen, write a lockword exit (or add to a current one), placing your value in SPAMODSI. There are two requirements you must meet:

1. The screen name cannot be longer than four characters.

2. There must be separate ones for each ADFID, since the transaction trailer (TRXTRLR) is appended to the 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.

CONVERSATIONAL SIGN-OFF EXIT PROCESSING

A user exit may be invoked at sign-off to a conversation (that is, the terminal operator enters 'C' on the Primary Option Menu). The user-written exit is invoked by IMSADF II Conversational Sign-off module, MFC1T99. The exit can be written in COBOL, PL/I or ASSEMBLER language. 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 layout of the SPA is included in IMSADF.MACLIB.ASM as member name SPACOBOL, SPAPLI, or SPAASM.

DBPCB is a data base PCB, added to the PSB ????T99 by the application developer. To add the PCB to the IMSADF II PSB, copy the PSB, ????T99, from library IMSADF.PSBLIB.SOURCE 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.

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    =   userid (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    =   userid (6 characters)
NEWPG       =   new project/group (2 characters)
```

NEWPG is the newly signed on project group when option 'F' is entered on the Primary Option Menu. NEWPG is two characters in length.

Register 15 is checked upon return to the termination module, MFC1T99. A value of 0 in Register 15 indicates on OK condition. A non-zero 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.

IMSADF II supplied a lockword exit for sign-on called MFC1E01 and one for sign-off (MFC1E99). Please note that the exits supplied by IMSADF II are 'dummy' exits, and that you are expected to replace them by yours when you code them. There should be no IBM maintenance to either of the modules which you are replacing. The IMS Application Development Facility II Version 2 Release 2 Installation Guide gives instructions for incorporating the exits into your IMSADF II system. Implementation of them takes two forms based on the language in which the exits are written.

When your exit is written in either COBOL or Assembler, it is incorporated into the IMSADF II system as an SMP Usermod, and maintenance on any modules which are part of ????T99, the Sign-off Processor, is accomplished without any manual intervention required.

If you write your exit in COBOL or assembler, the name of your sign-off exit is MFC1E99. Compile it and link it with any subroutines you require in IMSADF.RULLIB.

If your sign-off exit is written in PL/I, you are encouraged to give it the name MFC1E99. Your code will be a non-Main PL/I exit. When you use PL/I exits you must add a module for the interface to it. Since you are changing the link-edit structure for sign-off, you must effect your change outside of the SMP environment. Place the object text in a library of your choice, from which you will include it at link-edit time.

Alternative Sign-off Display Format

The sign-off exit may be used to control screen flow after processing for the current application is complete. A field called SPAMODSI contains the four-character name of the sign-on screen (MFS MOD name or BMS MAPSET name). If you wish a different screen to display following sign-off, write a lockword exit (or add to a current one), placing your value in SPAMODSI. There are two requirements you must meet:

1. The screen name cannot be longer than four characters.

2. There must be separate ones for each ADFID, since the transaction trailer (TRXTRLR) is appended to the 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.

USER AUDIT EXIT PROCESSING

A user exit is provided in the Auditor module (MFC1V14) to allow the addition of installation-dependent audit modules. A call will be executed to a user exit module for any audit descriptor code in the series 70 through 99 or W0 through Z9. This module, which is linked with a copy of the transaction driver, can perform specific audit/logic functions not handled by the Auditor.

The following parameters are passed from the Auditor to the user exit module:

- Addr (audited field)
- Addr (field definition in the Segment Layout Rule)
- Addr (audit operation code - two bytes, 70 to 99 or W0 to Z9)
- Addr (Audit Data Base PCB). See Note 1 on page 6-54.

- Addr (Common Segment Handler communication area (COMOPT))
- Addr (True/False indicator - 1 byte)
- Addr (function indicator - 1 byte)
- Addr (SPA area)
- Addr (List of user data base PCBs). See Note 1 on page 6-54.
- Addr (operation descriptor concatenated key COKEY)
- Addr (related field - if specified). See Note 2 on page 6-54.
- Addr (related field description in the Segment Layout Rule). See Note 2 on page 6-54.
- Addr (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)

Notes:

1. The tenth parameter (COKEY) contains the key of the Operation Descriptor that caused the exit routine to be invoked. If dynamic audit rules are used and a Data Descriptor is required, a GN is issued against the appropriate Data Descriptor ID as described below. This GN call will point to the Audit Data Base PCB address (parameter four) passed on the call to the exit. If a call is made to a user data base, the 'PCB' parameter can point to either the actual address of the PCB or the relative PCB number as explained in Special Processing. The actual address is in a list pointed to by parameter nine. The exit routine is responsible for determining which PCB address is required from the list passed.

2. If a related field is specified in the Related Field Name of the Operation Descriptor, it is addressed in these two parameters. If not, these two parameters will point to the audited field like parameters 1 and 2.

3. The parameter list passed from the auditor to the audit exit is changed to include the following addresses: 3 dummy positions for future use, the IOPCB, the ALTIOPCB, the EXPIOPCB, PCB count, and up to 120 user PCBs. This is consistent with the parameter list passed to special processing routines. The change eliminates the need for an assembler subroutine to access application data bases when the SEGHNDLR is not appropriate. Access to non-IMSADF II data base PCBs allows the use of IMSADF II for all applications to maintain a consistent interface for end users. This enhancement also allows the audit exit routines to issue DL/I calls to the IOPCB. It also provides access to the IOPCB date and time in standard processing without having to write special processing routines to attain access to the IOPCB. All these new parameters were added to the end of the old parameter list. Existing audit exits require no modification.

Since a user exit module is called for a wide range of descriptor codes, it should first check the descriptor code (third parameter passed) to determine which of the user audit routines should be invoked. This module then can call or branch to the desired audit routine.

The exit routine(s) can use the same coding conventions described under the Special Processing sections in this chapter. Data base I/O can be handled through calls to the Common Segment Handler (CALL SEGHNDLR). Data can be moved to and from the SPA area through calls to the Data Mapper (CALL MAPPER).

If the user audit routine requires Data Descriptor segments and the Audit Data Base is used, it should check the function indicator (1-character field) to determine which leg of the Audit Data Base is being retrieved. Depending on the value of the function indicator, the following calls should be made:

Function Indicator Value	Call
1 -	CALL SEGHNDLR('DA','GN ',COKEY,'FE',PCB,AREA) (Automatic Field Assignment Leg)
2 -	CALL SEGHNDLR('DF','GN ',COKEY,'FE',PCB,AREA) (Field Audit Leg)
3 -	CALL SEGHNDLR('DM','GN ',COKEY,'FE',PCB,AREA) (Message Sending Leg)

If an Audit Static Rule is used, the final parameter points to an area in the Auditor which will contain all the required Data Descriptors. This area is formatted as follows:

COUNT	DATA DESCRIPTORS

The COUNT field is a full word which contains the number of static descriptors which follow. If static rules are used and no descriptors are present, the count is zero. If dynamic rules are used the count is -1. The Data Descriptors are each 28 bytes in length and appear as if retrieved from the data base as follows:

```
0001(                )         0002(...etc.)
```

If the Data descriptors are in the spanned format each set of spanned records will be considered a static descriptor. They will appear in the area as follows:

```
0001 (up to 70 characters)         0002(...etc.)
```

Before returning to the Auditor, the user module must set the True/False indicator as follows:

```
True  x'80'
False x'00'
User set X'C0'
```

The user module may set a field in error with an error message number. This is accomplished with the SETERROR call. If this is done, the TRUE/FALSE indicator should be set to X'C0'. Upon return, the Auditor will terminate auditing on the current field if the indicator is set to X'C0'. If set to X'80' or X'00', the indicator will be used to determine whether the NEXT TRUE SEQ or NEXT FALSE SEQ should be used. Return codes are <u>not</u> checked by the Auditor upon return from the exit.

A user exit module is link-edited with a copy of a transaction mini-driver with which it is associated. The Rules Generator performs all link-edits of conversational, nonconversational, and batch transaction driver and will include the appropriate exit module. The Rules Generator parameters AEXIT=(modname) and ALANG=(language) should be included on the GENERATE statement which performs the link-edit.

HOST SYSTEM SERVICES

A user written routine (exit or special processing routine) sometimes requires some host system services; that is, CICS system services in a CICS/OS/VS environment and OS system services in an IMS/VS environment. The assembler macro instructions described in this section are supplied as members of IMSADF.ADFMAC and may be used in a user written routine to request system services - invoke an equivalent host system (CICS or OS) service request. Under CICS/OS/VS, a CICS Command Level service request is invoked; under IMS/VS, an OS service request is invoked. For example, ADFGMAIN invokes a CICS EXEC GETMAIN under CICS/OS/VS and invokes an OS GETMAIN under IMS/VS. In this way, a user written routine can run in either a CICS/OS/VS or an IMS/VS environment without change.

The result of an IMSADF II system service request is normally the same in either a CICS/OS/VS or IMS/VS environment, but is not always the same. For example, when storage is not available for an ADFGMAIN, a CICS/OS/VS task is suspended, but an IMS/VS transaction is abnormally terminated. The IMSADF II system service routines are written to provide analogous, but not identical services.

On entry to any user written routine, register 11 always points to the address of an IMSADF II control block (MFC1CAS). The expansion of each of the IMSADF II system service macro instructions references register 11, using its address as the anchor into the IMSADF II system. Each macro expands into a call (BALR R14,R15) to the appropriate IMSADF II system service routine.

A DSECT is generated by the expansion of the ADFCAS macro, and this macro must be included in every routine that includes an IMSADF II system service macro.

```
CAS      ADFCAS TYPE=DSECT
```

For some user written routines, it may be necessary to issue CICS command, using the CICS Command Level application program interface. When such commands are issued, CICS expects register 13 to contain the address of DFHEISTG, a control block passed to IMSADF II by CICS. On entry to any user written routine, register 13 always contains the address of a register save area, but it is not the address of DFHEISTG passed by CICS. On entry to a user written routine register 11 always contains the address of the address of MFC1CAS, an IMSADF II control block which contains the address of the DFHEISTG passed by CICS.

The following example shows an Assembler Language user written routine issuing a CICS command.

```
         .
         .
         .
         ST    R13,R13SAVE         SAVE REG 13
         L     R13,0(,R11)
         L     R13,CASEISTG-CAS(,R13)
*
         EXEC CICS HANDLE CONDITION INVREQ(EHCRTN)
*
         L     R13,R13SAVE         RESTORE REG 13
*
         .
         .
         .
CAS      ADFCAS TYPE=DSECT
```

Refer to "Programming Techniques and Restrictions" in the CICS Application Programmer's Reference Manual (Command Level) for tips useful in coding IMSADF II user written routines.

MACRO INSTRUCTIONS

In the description of macro instructions, the following general rules apply:

- If a parameter is written in upper case letters, code the parameter exactly as shown. (example: DUMP or RET=USE)
- If a parameter is written in lower case letters, substitute the indicated value, address, or label. (example: value or label)
- If a parameter is written in a combination of upper case and lower case letters separated by an equal sign, code the upper case letters and equal sign as shown, and substitute the indicated value, address, or label. (example: EPLOC=addr)
- Where registers are specified in place of symbols or values, the register value is enclosed in parentheses.

Figure 6-5 presents a sample macro instruction, ADFSAMP, and summarizes all the coding information that is available for it.

This figure is divided into three columns.

- The first column contains those parameters that are required for this macro instruction. When two or more lines appear together, the parameter appearing on one and only one of the lines must be coded.
- The second column contains those parameters that are optional for this macro instruction. When two or more lines appear together, the parameter appearing on one and only one of the lines may be coded if desired. The default is underscored.
- The third column provides additional information for coding the macro instruction.

```
    label           symbol.  Begin in column 1.

b                   one or more blanks must precede ADFSAMP.

ADFSAMP

b                   one or more blanks must follow ADFSAMP.

R
RC
RUC

,DATA=value         symbol or decimal digit.

    ,DCB=(1)
    ,DCB=addr       RX-type address or (1) or (2) - (12).

    ,FMT=HEX
    ,FMT=DEC
    ,FMT=BIN
```

Figure 6-5. ADFSAMP Macro

ADFABEND macro

The ADFABEND macro instruction initiates abnormal termination of the task.

```
    label          symbol.  Begin in column 1.

b                  one or more blanks must precede ADFABEND.

ADFABEND

b                  one or more blanks must follow ADFABEND.

comp code          symbol, decimal or hex digit,
                   or (0) or (2) - (12).

    ,DUMP
```

Figure 6-6. ADFABEND Macro

comp code
specifies the completion code associated with the abnormal termination. The accepted value range is 0 - 4095.

DUMP
specifies that a dump is requested of virtual storage areas assigned to the task and control blocks pertaining to the task.

ADFBLDL macro

The ADFBLDL macro instruction completes a list of information from the directory of a partitioned data set (PDS). The address of a BLDL list must be supplied. Each entry in the the list must be at least 60 bytes. Any number of entries may be specified in the list.

A completion code is returned in register 15, a non-zero return code indicating the request could not be satisfied. The most likely cause of a non-zero return code (R15=4) is that the referenced PDS does not contain one of the requested entry names.

```
    label           symbol.  Begin in column 1.

b                   one or more blanks must precede ADFBLDL.

ADFBLDL

b                   one or more blanks must follow ADFBLDL.

dcb addr            RX-type address or (1) or (2) - (12).
0       decimal digit

,list addr          RX-type address or (0) or (2) - (12).
```

Figure 6-7. ADFBLDL Macro

dcb addr
specifies the address of the data control block (DCB) for an open partitioned data set (PDS), or zero can be specified to indicate that the data set is in a job library, step library, or link library.

list addr
specifies the address of a BLDL list.

ADFDLET macro

The ADFDLET macro instruction cancels the effect of a previous ADFLOAD macro instruction.

```
    label          symbol.  Begin in column 1.

b                  one or more blanks must precede ADFDLET.

ADFDLET

b                  one or more blanks must follow ADFDLET.

EPLOC=addr         RX-type address or (0) or (2) - (12).
DE=addr            RX-type address or (0) or (2) - (12).
```

Figure 6-8. ADFDLET Macro

EPLOC
specifies the address of an eight-character name, padded on the right with blanks if necessary.

DE
specifies the address of the name field in a 60-byte list entry that was constructed using the ADFBLDL macro instruction.

ADFFMAIN macro

The ADFFMAIN macro instruction releases an area of virtual storage previous allocated via an ADFGMAIN instruction.

```
    label           symbol.  Begin in column 1.

b                   one or more blanks must precede ADFFMAIN.

ADFFMAIN

b                   one or more blanks must follow ADFFMAIN.

R

LV=value            symbol, decimal digit, or (0) or (2) - (12).

A=addr              A-type address or (1) - (12).
```

Figure 6-9. ADFFMAIN Macro

R
required.

LV
specifies the length in bytes of the virtual storage to be released.

A
specifies the address of the virtual storage to be released. Must be on a double word boundary.

ADFGMAIN macro

The ADFGMAIN macro instruction allocates virtual storage. The address is returned in register 1.

```
    label           symbol.  Begin in column 1.

b                   one or more blanks must precede ADFGMAIN.

ADFGMAIN

b                   one or more blanks must follow ADFGMAIN.

R

LV=value            symbol, decimal digit, or (0) or (2) - (12).
```

Figure 6-10. ADFGMAIN Macro

R
required.

LV
specifies the length, in bytes, of the requested virtual storage. The number should be a multiple of eight; if it is not, the length allocated will be the next higher multiple of eight.

ADFLOAD macro

The ADFLOAD macro instruction is used to bring the load module containing the specified entry name into virtual storage, if a usable copy is not already available in virtual storage. The address of the entry point is returned in register 0.

If the specified entry name cannot be located, the task is abnormally terminated.

```
    label           symbol.  Begin in column 1.

b                   one or more blanks must precede ADFLOAD.

ADFLOAD

b                   one or more blanks must follow ADFLOAD.

EPLOC=addr          RX-type address or (0) or (2) - (12).
DE=addr             RX-type address or (2) - (12).

    ,DCB=0
    ,DCB=addr       RX-type address or (1) or (2) - (12).
```

Figure 6-11. ADFLOAD Macro

EPLOC
specifies the address of an eight-character name, padded on the right with blanks if necessary.

DE
specifies the address of the name field in a 60-byte list entry that was constructed using the ADFBLDL macro instruction.

DCB
specifies the address of the data control block (DCB) for the partitioned data set (PDS) to be searched.

This parameter is ignored under CICS.

IF DCB=0 is specified, the data sets referred to by either the STEPLIB or JOBLIB DD statement are first searched for the entry name. If the entry name is not found, the link library is searched.

CHAPTER 7. IMSADF II UNDER IMS/VS

DEFINING THE APPLCTN AND TRANSACT MACROS

The IMS/VS APPLCTN and TRANSACT macros define the application programs to be scheduled and the transactions to be processed.

Figure 7-1 shows IMSADF II modules, naming conventions, and IMS/VS system definition for conversational, nonconversational, and batch transaction drivers.

IMS/VS TRANSACTION (CONVERSATIONAL) PROCESSING

PROCESSING	LINK-EDIT NAME	PSB NAME	TRANSACTION NAME
STANDARD SEGMENT and TEXT UTILITY	ssssTtxa(1)	ssssTtxa	ssssTtxa(2)
CLUSTERED STANDARD SEGMENT and SPECIAL PROCESSING	ssssTtxa(1) ssssZxxa(1)	ssssTtxa	ssssTtxa(2)
SPECIAL PROCESSING	ssssVxxa(1)	ssssVxxa	ssssVxxa(3)

IMS/VS TRANSACTION (NONCONVERSATIONAL) PROCESSING

STANDARD SEGMENT	ssssTbtx(1)	ssssTbtx	ssssTbtx(2)
SPECIAL PROCESSING	ssssTbxx(1)	ssssTbxx	ssssTbxx(3)

NOTES:

```
ssss = application system ID
  tx = cluster code to identify PSB and TRANSACT
   T = transaction processing (conversational or
        nonconversational)
   V = special processing transaction
  xx = special processing transaction ID
   Z = special processing module name
   1 = Created by GENERATE OPTION=STLE,NCLE,SPLE or TPLE
        and PGMID=tx or xx
   2 = SOMTX operand must match tx
   3 = TRXID operand must match xx for CVALL and TPALL generations
        when SPECIAL=YES
   a = Transaction Trailer (TRXTRLR) specified at installation
        or blank if Transaction Trailer not specified
   b = Transaction Trailer (TRXTRLR) specified at installation
        or zero if Transaction Trailer not specified
```

Figure 7-1 (Part 1 of 2). Modules, Naming Convention, and IMS/VS System Definition

IMS/VS BMP PROCESSING

PROCESSING	LINK-EDIT NAME	PSB NAME	TRANSACTION NAME
STANDARD SEGMENT	ssssTbtx(2)	ssssTbtx	ssssTbtx
SPECIAL PROCESSING	ssssTbxx(2)	ssssTbxx	ssssTbxx
STANDARD or SPECIAL PROCESSING	ssssBDrr(3)	ssssBDrr	(1)

IMS/VS BATCH PROCESSING

STANDARD SEGMENT or SPECIAL PROCESSING	ssssBDrr	ssssBDrr	(1)

NOTES:

```
ssss = application system ID
   T = constant T
  xx = special processing transaction ID
  rr = Batch Driver Rule identification
  tx = cluster code to identify PSB
   1 = not specified
   2 = Created by GENERATE OPTION=NCLE,PGMID=tx
         or OPTION=TPLE,PGMID=xx
   3 = Created by GENERATE OPTION=BDLE, PGMID=rr,...
  BD = Batch Transaction Driver
   b = Transaction Trailer (TRXTRLR) specified at installation
         or zero if Transaction Trailer not specified
```

Figure 7-1 (Part 2 of 2). Modules, Naming Convention, and IMS/VS System Definition

CONVERSATIONAL PROCESSING

An APPLCTN and TRANSACT macro pair will exist for each transaction cluster code (SOMTX=tx) identified in the appropriate Secondary Option Menu Rule. The cluster code provides a means to relate Standard Processing and clustered special processing IDs. The IDs are considered related if they access the same Program Specification Block (PSB). In addition, an APPLCTN and TRANSACT macro pair must be specified for each non-clustered Special Processing identification in the Secondary Option Menu Rule for Special Processing. For each APPLCTN and TRANSACT macro pair, the link-edited application program name, PSB name, and transaction name will be identical. Refer to Figure 7-1 for a description of the naming convention.

The Rules Generator performs the link-edits for the various transaction drivers. The use of Special Processing with a Transaction Driver requires that the Special Processing Routine be link-edited with a conversational mini-driver. Therefore, a load module must exist for each Special Processing identification.

The following are examples of the APPLCTN and TRANSACT macro pairs that are defined to IMS/VS System Generation for the Conversational

Transaction Driver. SPA=28 is specified in the following examples. For a description of the SPA parameter, see the IMS Application Development Facility II Version 2 Release 2 Installation Guide.

Example: Conversational Driver (Standard Processing, Text Utility, clustered Special Processing).

The transaction cluster code is CT.

```
APPLCTN     PSB=MFC1TCT,PGMTYPE=TP
TRANSACT    CODE=MFC1TCT,MSGTYPE=(MULTSEG,RESPONSE,01),      X
            INQ=NO,MODE=SNGL,EDIT=UC,SPA=28
```

Example: Conversational Driver (Special Processing).

Department Overtime (DO) is a special processing identification. There will be one APPLCTN, TRANSACT macro pair for each non-clustered Special Processing identification.

```
APPLCTN     PSB=MFC1VDO,PGMTYPE=TP
TRANSACT    CODE=MFC1VDO,MSGTYPE=(MULTSEG,RESPONSE,01),      X
            INQ=NO,MODE=SNGL,EDIT=UC,SPA=28
```

Note: If a Transaction Trailer was specified at installation, the names specified in the PSB= and CODE= operands would contain the Transaction Trailer character in the eighth byte.

NONCONVERSATIONAL PROCESSING

An APPLCTN and TRANSACT macro pair are specified for each Special Processing ID and for each group of related Standard Processing IDs. Standard Processing IDs are considered related if they access the same Program Specification Block (PSB).

Example: Online Driver (Standard Processing).

The application, MFC1T0CT, will process each of the related Standard Processing IDs that can be accessed from PSB, MFC1T0CT. The Input Transaction Rule must indicate whether or not a reply is required. In this example, the keyboard will not be locked after data entry.

```
APPLCTN     PSB=MFC1T0CT,PGMTYPE=TP
TRANSACT    CODE=MFC1T0CT,MSGTYPE=(SNGLSEG,NONRESPONSE,01),  X
            MODE=SNGL
```

Example: Online Driver (Standard Processing).

The application, MFC1T0CR, will process each of the related Standard Processing IDs that can be accessed from PSB, MFC1T0CR. For this example, the Input Transaction Rules must indicate a reply is to be inserted to the IOPCB. MSGTYPE of RESPONSE indicates the keyboard will be locked until the response is sent.

```
APPLCTN     PSB=MFC1T0CR,PGMTYPE=TP
TRANSACT    CODE=MFC1T0CR,MSGTYPE=(SNGLSEG,RESPONSE,01),     X
            MODE=SNGL
```

Example: Online Driver (Special Processing).

The application, SAMPTOST, will process the Special Processing ID, ST. If the Special Processing Input Transaction Rule indicates a reply is to be sent to the entering terminal, the MSGTYPE on the TRANSACT macro may be RESPONSE or NONRESPONSE. If the Input Transaction Rule indicates no reply is to be ISRTed to the IOPCB, then MSGTYPE must be NONRESPONSE.

```
APPLCTN     PSB=SAMPTOST,PGMTYPE=TP
TRANSACT    CODE=SAMPTOST,MODE=SNGL,                          X
            MSGTYPE=(SNGLSEG,NONRESPONSE,01)
```

Note: If a Transaction Trailer was specified at installation, the names specified in the PSB= and CODE= operand would contain the Transaction Trailer character in the sixth byte instead of zero.

BMP PROCESSING

The Online Transaction Driver (nonconversational processing) or the Batch Transaction Driver is available to implement BMP processing. The former is used if the BMP requires the IMS/VS message queue. Otherwise, the batch transaction driver is used. IMS/VS does not require the PSB name to match the application program name in BMP processing. However, it is recommended that they match for simplification.

Examples of BMP APPLCTN and TRANSACT macros follow.

Example: BMP (Online Driver)

This APPLCTN and TRANSACT macro pair define the Special Processing application DC to IMS/VS.

```
APPLCTN     PSB=MFC1TODC,PGMTYPE=BATCH
TRANSACT    CODE=MFC1TODC,MSGTYPE=(SNGLSEG,NONRESPONSE,01),   X
            INQ=NO,MODE=SNGL,EDIT=UC,PRTY=(0,0,65535)
```

Note: If a Transaction Trailer was specified at installation, the names specified in the PSB= and CODE= operands would contain the Transaction Trailer character in the sixth byte instead of zero.

Example: BMP (Batch Driver)

```
APPLCTN     PSB=MFC1BDLF,PGMTYPE=BATCH
```

BATCH PROCESSING

Batch processing application programs are not specified in the APPLCTN macro. IMSADF II batch processing is implemented using the batch batch transaction driver, link-edited with a Batch Driver Rule. In this way, the generalized batch transaction driver takes on application uniqueness.

DEFINING THE PSBS FOR IMSADF II APPLICATIONS

A PSB must be generated for each IMS/VS transaction. If the IMSADF II dynamic rule data bases are DL/I, they must be in the PSB to accommodate such functions as auditing, automatic message sending, sign-on, secondary transactions, for the Conversational, Online, and Batch Transaction Drivers. The application data base PCBs for these drivers follow the IMSADF II dynamic rule data base PCBs. An alternate IOPCB and express IOPCB must also be specified to precede IMSADF II data bases. Each of the three transaction drivers also has unique requirements which are described later in this section.

If the 'data compare before update' facility is being used for conversational transactions, the PSBGEN statement must contain a MAXQ parameter. A 'Q' command code will be used for each path/segment that requires compare before update. A MAXQ=25 value should be adequate for most PSBs, but should be reviewed for each IMSADF II conversational transaction.

Included in the PSB source library are Assembler macros BUILDT07 and BUILDT08, and a COPY member, BUILDT09. Add your application PCBs and name the PSB appropriately for your application. Note that BUILDT08 requires a 3270 printer to be identified.

The coding convention for these macros is:

```
label       BUILDT07    ,                    comments
                or
label       BUILDT08    LTERM=nnnnnnnn  ,    comments
                or
            COPY BUILDT09                    comments
                or
            BUILDT9B
```

(label may be specified, but will be ignored.)

LTERM: (Valid only with BUILDT08). The name of an LTERM (usually a printer) that will receive any error messages produced by a nonconversational application that was defined as not generating a reply to input (OPTIONS=TPIT).

The specification of the Building Block must be the first non-comment in the input stream. It is followed by the PCB and SENSEG statements for the application data bases, which in turn are followed by a PSBGEN and END statement.

An example is the PSB for the OR cluster code for the Sample Program Standard Processing application.

```
BUILDT07
PCB     TYPE=DB,DBDNAME=DI21PART,PROCOPT=AP,KEYLEN=43
SENSEG  NAME=PARTROOT
SENSEG  NAME=STANINFO,PARENT=PARTROOT
SENSEG  NAME=STOKSTAT,PARENT=PARTROOT
SENSEG  NAME=CYCCOUNT,PARENT=STOKSTAT
SENSEG  NAME=BACKORDR,PARENT=STOKSTAT
PSBGEN  LANG=ASSEM,PSBNAME=SAMPTOR
END
```

The IMSADF II PSB source library and the IMSADF II assembler macro library must be a part of the SYSLIB concatenation in the PSBGEN PROC.

The contents of the building blocks are derived from the parameters specified during the installation of an IMSADF II system. The GEN=NO keyword may be coded to provide documentation of the building blocks. The following assembler input stream is an example of documenting a Nonconversational building block that contains a 3270 printer LTERM name of RPRT1:

```
BUILDT08    LTERM=RPRT1,GEN=NO
END
```

VSAM FILES

You may access a VSAM KSDS data set using IMSADF II in an IMS/VS environment. To do this, perform a DBDGEN to define the file to IMS/VS as a SHISAM root-only DL/I data base. Access to the file can only be made via the primary key. Records can be retrieved, added, updated, and deleted using normal IMSADF II functions.

The TYPE operand of the SEGMENT statement is used in the Rules Generator to define a VSAM file:

TYPE= KSDS for Key Sequenced Data set

PARENT= 0

NAME= the SEGM NAME as used in the DBDGEN

See also "VSAM Calls" on page 3-19 in Chapter 3, "Audit Language."

Note: To be used with IMSADF II, a VSAM data set must contain at least one record.

DEFINING BATCH TRANSACTION DRIVER JCL

The Batch Transaction Driver is executed as an IMS/VS batch job or as an IMS/VS BMP. You may use IMS procedure DLIBATCH, DBBBATCH or IMSBATCH as described in the IMS/VS System Programming Reference Manual appending the following JCL statements as appropriate.

PRINTER	DD	Defines the hard copy output from the batch run. DCB parameters supplied by the Batch Transaction Driver are RECFM=FBA and LRECL=133. BLKSIZE must be specified by the user.
RSTRTIN	DD	Defines the sequential input data set that is to contain the restart (RSTRT) transaction if a batch job is being restarted. DD DUMMY can be specified if the job is a normal run. DCB parameters are RECFM=F and LRECL=80.
TRANSIN	DD	Defines the sequential input data set that is to contain the transaction messages to be processed. There are no default values for the DCB parameters BLKSIZE, LRECL, and REFCM. Therefore, they must be specified. RECFM=F, FB, V, or VB LRECL=8 to 255
TRANSOUT	DD	(Optional) Defines the sequential output data set that is to contain the unprocessed transaction messages if STOP has been entered by the OS/VS System Operator. If omitted, the WTOR message to the OS/VS System Operator is not issued. The DCB parameters are the same as those for TRANSIN.
ERRMSG	DD	(Optional) Defines the hard copy error message data set. All data base transactions that are in error and their associated error messages may be logged here. DCB parameters specified by the Batch Transaction Driver are RECFM=FBA and LRECL=133. BLKSIZE must be specified by the user.
ERRTRX	DD	(Optional) Defines a data set to which error transactions may be written. The DCB parameters must be the same as those specified for TRANSIN.
SECTRX	DD	(Optional) Defines a sequential data set to receive secondary transaction records when executing batch IMS/VS. See "Secondary Transactions/Output Message Routing" on page 7-10 for additional information. DCB parameters that must be specified by the user are

LRECL, BLKSIZE, and RECFM. The largest LRECL is 2048 bytes and RECFM is FB or VB.

CARDOUT DD Punched output of the batch driver for static audit load module creation. The Audit Data Base is read and the batch driver creates output in this DDNAME which must be **input** to the Rules Generator (which actually creates the static audit load module).

EXAMPLE:

```
//BATCH    JOB   ACCNT,NAME,MSGLEVEL=1
//         EXEC  DLIBATCH,MBR=MFC1BD01,RGN=1536K
//PRINTER  DD    SYSOUT=A
//RSTRTIN  DD    DUMMY
//TRANSIN  DD    DSN=USER.TRANSIN(MEMBER1),DISP=SHR
//*
//*      DATA BASE DD STATEMENTS
//*
//DB1      DD    DSN=USER.DB1,DISP=OLD
//DB2      DD    DSN=USER.DB2,DISP=OLD
```

An alternate batch driver procedure is found in IMSADF.JCLLIB(PROCB). During IMSADF II installation it is copied to your procedure library as ????B (where ???? = ADFID).

CONVERSATIONAL TRANSACTION CLUSTERING

STANDARD PROCESSING TRANSACTIONS

One or more standard processing Conversational transactions can be processed by the same IMS/VS transaction code. This feature of Conversational processing provides the capability for user transactions to be clustered when they access common data bases, when they are part of the same application, or when they can provide better performance because of reduced IMS/VS transaction switching. This clustering is controlled by the Secondary Option Menu rule, which lists all transaction IDs (TRXIDs) valid for an application system and their associated IMS/VS cluster code. (Refer to the SOMTX and SYSID operands and the SOM GENERATE OPTION.) The application system ID and cluster code determine the IMS/VS transaction and PSB under which a TRXID is scheduled and the link-edit name given to the executable conversational driver. That name is ssssTtx, where ssss is the application system ID, T is a constant, and tx is the cluster code. Following is an example of the Rules Generator statement to link-edit a Standard Processing Conversational Driver.

```
GENERATE OPTIONS=STLE,PGMID=tx
```

will produce a driver that can handle standard processing Conversational transactions that specify a cluster code (SOMTX) equal to the PGMID of tx. The driver name will be ssssTtx. Any audit exit or DL/I exit will be link-edited in the load module ssssTtx.

SPECIAL PROCESSING TRANSACTIONS

Special Processing transactions can be clustered under a Standard Processing transaction (ssssTtx) or can require one IMS/VS transaction (ssssVxx) per IMSADF II Special Processing transaction ID (TRXID). To cluster a Special Processing transaction, the GENERATE statement for the Conversational Input Transaction Rule specifies a cluster code (SOMTX) which matches the program ID (PGMID) on the Conversational Driver LINK-EDIT (STLE) and DYNLOAD=YES. Additionally, the STLE LINK-EDIT must specify the operand DYNLOAD=YES to indicate that any Special Processing Routines will be dynamically linked at execution time. To accomplish the link-edit of the Special Processing Routine, the GENERATE option SPLE is performed for each Special Processing Routine with DYNLOAD=YES. Any Audit Exit or DL/I Exit specifications must be associated with the STLE LINK-EDIT. MAPTABLE and SHTABLE specifications can be associated with individual SPLE GENERATEs or with the clustered STLE GENERATE to incorporate all clustered SPR requirements for MAPTABLE and SHTABLE.

Following is an example of one Standard Processing and two clustered Special Processing transaction IDs, all executed under one Standard Processing IMS/VS transaction.

```
SYSTEM     SYSID=SAMP,SOMTX=01,PGROUP=ZZ

.
.
.
GENERATE   OPTIONS=CVALL,DBPATH=(B1,B2),TRXNAME='STANDARD TRXID A1',
           TRXID=A1
GENERATE   OPTIONS=CVALL,DBPATH=(X1),TSEGS=(P1),DYNLOAD=YES,
           TRXNAME='SPECIAL TRXID A2',TRXID=A2,SPECIAL=YES
```

DYNLOAD indicates that A2 will be clustered under SAMPT01

```
GENERATE   OPTIONS=CVALL,DBPATH=(X2),TSEGS=(P2),DYNLOAD=YES,
           TRXNAME='SPECIAL TRXID A3',TRXID=A3,SPECIAL=YES
```

DYNLOAD indicates that A3 will be clustered under SAMPT01

```
GENERATE   OPTIONS=SOM
```

The Secondary Option Menu Rule will incorporate TRXIDs A1, A2, A3 and will indicate that A2 and A3 are clustered under the SOMTX of 01. Note, if A2 and A3 should be clustered under another SOMTX, the SOMTX value would be specified on the CVALL GENERATE.

```
GENERATE   OPTIONS=STLE,PGMID=01,AEXIT=AUDITPGM,
           DEXIT=DLIPGM,MAPTABLE=(AA,AB),SHTABLE=(X1,X2),
           DYNLOAD=YES
```

The above GENERATE builds a load module SAMPT01 that processes Standard Processing TRXIDs with the cluster code (SOMTX) of 01, and will dynamically LINK at execution time Special Processing TRXIDs clustered with SOMTX=01.

Any audit or DL/I Exit will be link-edited with the mini-driver. If MAPTABLE or SHTABLE is utilized by the audit or DL/I Exit or by a clustered Special Processing Routine, the STLE LINK-EDIT can incorporate all the requirements. Note that MAPTABLE and SHTABLE are optional, since Mapping Rules or Segment Handler Rules will be found in the PRELOAD list, or the Composite Load Module or dynamically loaded, depending on the preload or composite load module options selected.

```
GENERATE   OPTIONS=SPLE,PGMID=A2,DYNLOAD=YES
```

The above GENERATE will generate a load module SAMPZA2. When TRXID,A2, is selected for processing by a terminal operator, it will be scheduled under the IMS/VS transaction SAMPT01. The mini-driver associated with SAMPT01 will dynamically LINK to SAMPZA2. It will remain in the Message Region until there is a request for another SPR to be dynamically linked. DEXIT and AEXIT do not apply to the SPLE LINK-EDIT when DYNLOAD=YES. MAPTABLE and SHTABLE may be specified and will be utilized at execution time only if the clustered STLE GENERATE has no MAPTABLE and SHTABLE specification. Otherwise, the MAPTABLE and SHTABLE on the STLE GENERATE takes priority.

```
GENERATE   OPTIONS=SPLE,PGMID=A3,DYNLOAD=YES
```

The above GENERATE will generate a load module SAMPZA3. Logic for IMS/VS scheduling and dynamic LINK are the same as for TRXID A2.

RUNNING AN IMSADF II TRANSACTION UNDER MULTIPLE IMS/VS TRANCODES

The standard IMSADF II clustering algorithm allows multiple IMSADF II transactions to execute under one IMS/VS transaction (TRANCODE). This clustering is done using the Rules Generator parameters SOMTX and PGMID.

However, it may become necessary to run a single IMSADF II conversational transaction under multiple IMS/VS TRANCODEs. This might be done for scheduling considerations or to use multiple PSBs. For example, one IMSADF II transaction might be used to update multiple mirror-image data bases which must remain physically separated.

IMSADF II allows a user program to determine dynamically which TRANCODE should be scheduled and therefore which PSB should be used. This **user-clustering** is accomplished by storing a two-character user cluster code (no blanks) into the SPA field **SPACLUS** using a user-written Sign-on Exit, Audit Exit, or Special Processing Routine. The next time IMSADF II builds a new IMS/VS conversational TRANCODE for scheduling (after selection of an IMSADF II transaction during sign-on or following a transaction switch), this user cluster code will overlay positions 3 and 4 of the normal TRANCODE.

For example, the following Rules Generator inputs will produce an IMSADF II transaction "PA" which normally runs under the IMS/VS TRANCODE "SAMPTOR".

```
  SYSTEM   SYSID=SAMP,
           SHEADING='SAMPLE PROBLEM',
           PGROUP=ZZ,
           SOMTX=OR
*
  SEGMENT  PARENT=0,ID=PA,NAME=PARTROOT,LENGTH=50,
           SNAME='PART SEGMENT'
    FIELD  ID=KEY,LENGTH=17,POS=1,KEY=YES,NAME=PARTKEY,
           SNAME='PART NUMBER'
    FIELD  ID=DESC,LENGTH=20,POS=27,SNAME='DESCRIPTION'
*
  GENERATE TRXID=PA,DBPATH=PA,
           TRXNAME='PA MAINTENANCE',
           OPTION=CVALL
```

The following Rules Generator statement will produce the "SAMPTOR" member (Mini-Driver):

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

However, based on USERID, LOCKWORD, or other information, the user exit might set SPACLUS to "4C", so that the scheduled TRANCODE would be "SA4CTOR". The following Rules Generator inputs will produce the member "SA4CTOR" if it must be a separate member.

```
  SYSTEM   SYSID=SA4C,
           SOMTX=OR
*
  GENERATE OPT=STLE,PGMID=OR
```

If the name "SA4CTOR" may be an alias for "SAMPTOR", the following Rules Generator statements would be used to build "SAMPTOR" with the alias "SA4CTOR".

```
  SYSTEM   SYSID=SAMP,
           SOMTX=OR
*
  GENERATE OPT=STLE,PGMID=OR,
           ALIAS=SA4C
```

As in the case of standard clustering, you must define each user-specified TRANCODE to the IMS/VS system. To nullify this user-clustering feature, the user program should set SPACLUS to binary 0.

SECONDARY TRANSACTIONS/OUTPUT MESSAGE ROUTING

When indicated in the Input Transaction Rule, the transaction driver ISRTs a message defined by the Output Format Rule (OFR) to a destination defined in the Message Data Base. The destination may be a program (secondary transaction) or one or more terminals (message route). If the destination is a program, the Message Data Base contains the IMS/VS transaction code for the program. The secondary transaction can be a non-Facility application program or an IMSADF II nonconversational transaction driver. The IMS/VS TRANSACT and APPLCTN macros define the secondary transaction to receive the message.

To generate a secondary transaction or alternate output message, the conditions for generation, the format of the output, and the destination must be specified using a combination of Rules Generator statements and Message Data Base and Audit Data Base entries. The Input Transaction Rule lists the secondary transactions that may be generated and the conditions under which they will be generated. The conditions for generation include the processing mode (0 to 6) and the processing completion (OKAY or ERROR). Field auditing also can be used to determine secondary transaction generation. When a field meets a specified set of audit criteria, the secondary transaction operation descriptor, 08, can be specified to indicate which secondary transaction is to be generated. (Refer to Chapter 5, "Audit Logic Processing"). If this Input Transaction Rule is for Special Processing, a SPA flag, SPASECTX, is available to the Special Processing Routine to tell the Transaction Driver whether or not to invoke secondary transaction generation and for what conditions the Driver should check. To incorporate secondary transaction processing the Input Transaction Rule, the STX operand(s) should be specified on the GENERATE statement.

The format of the STX operand is:

```
STX=( MFS ,xx [,OK][,ER] ,mode)
      TRX
```

where,

MFS indicates that the transaction message is to be inserted (ISRT) with a modname. The modname is also the Output Format Rule (OFR) name.

TRX indicates that the transaction message is a data string and is not ISRTed with a modname. Subsequent applications will receive the modname associated with the program creating the secondary transaction.

xx specifies the Output Format Rule ID. The OFR defines the layout of the message. The xx is expanded by the Rules Generator to become an eight-character OFR name. The expansion is ssORxxO1, where:

ss = first two characters of the application System ID
OR = constant 'OR'
xx = Output Format Rule ID
O1 = constant 'O1'

OK indicates that this transaction will be generated if the processing (data base update) was successful.

ER indicates that this transaction will be generated if the processing (data base update) was unsuccessful.

mode indicates the current processing mode (1 to 6) for which this secondary transaction should be generated. A value of 0 indicates any mode 1 to 6.

If field auditing is to determine whether the secondary transaction is generated, neither OK nor ER need be specified.

Refer to Figure 7-2 for a summary of the logic among Driver processing, STX operand values, and generating secondary transactions.

STANDARD PROCESSING		
Facility Driver	STX Operand	Generate Secondary
• Processing successful	If mode matches and 1. OK is specified or 2. Audit was satisfied	 Yes Yes
• Processing unsuccessful	If mode matches and 1. ER is specified or 2. Audit was satisfied	 Yes Yes

SPECIAL PROCESSING		
Facility Driver	STX Operand	Generate Secondary
• Special Processing routine sets *SPASECTX=1 or 3	If mode matches and 1. OK is specified or 2. Audit was satisfied	 Yes Yes
• Special Processing routine sets *SPASECTX=2 or 3	If mode matches and 1. ER is specified or 2. Audit was satisfied	 Yes Yes

Figure 7-2. Conditions for Generating Secondary Transactions

Note: *SPASECTX is a SPA flag available to the Special Processing routine to indicate to the Transaction Driver when to check for secondary transaction generation. A value of 1 or 3 indicates that a secondary transaction will be generated for each STX operand specifying OK. A value of 2 or 3 indicates that a secondary transaction will be generated for each STX operand specifying ER. In either case, the currently processed mode must match the mode in the STX operand before the secondary transaction can be generated. If SPASECTX is set to 0 upon return from the Special Processing Routine, the Driver does not invoke secondary transaction generation.

OUTPUT FORMAT

The format of the output message is defined by the Output Format Rule. There must be an Output Format Rule for each STX operand specified. The Output Format Rule itself is generated by SEGMENT and FIELD statements that define the layout of the message, which is a maximum of 1600 bytes. The operands TYPE=OUT and ID=XX, (where XX matches the XX of the STX operand) are specified on the SEGMENT statement. These statements will define the layout of the secondary transaction or alternate output message to be generated. Therefore, it must match the layout expected by the program scheduled to process the transaction or the MFS MOD used for output to an LTERM. If this is a secondary transaction to be processed by the Nonconversational Driver, the Output Format Rule must have the first three fields defined as follows: The eight-character IMS/VS transaction name followed by a blank, and the mode and the segment ID or Special Processing ID to be processed. (Refer to Section C of the example in this section.) The contents of the message comes from three sources - field data from segments current processed, text data from the Output Format Rule, and keyword data from the SPA or segment handler communication area. Refer to the SEGMENT statement, operand TYPE=OUT, for more detail.

MESSAGE ROUTING

To determine the destination name (LTERM or transaction), IMSADF II accesses the Message Data Base as described in "Message Data Base" on page 4-12. The key is the Output Format Rule name. The message may be routed to up to eight specified terminals.

A transaction can also be directed to a Multiple System Couple (MSC) link by using the Audit DIRECT operation. This operation causes the Output Format name and MSC name to be stored in a table during auditing and to be retrieved during secondary transaction routing. A search is made of the table on Output Format name. If found, the corresponding MSC link name is used as the destination.

A batch transaction driver, executing batch IMS/VS, writes the message to the sequential data set, identified by the DDNAME SECTRX. The SECTRX DCB parameters, RECFM=FB or VB, LRECL=n, and BLOCKSIZE=n must be specified. The maximum length for LRECL is 2K (2048 bytes), minus 4 bytes used by the drivers for control information. If the output message built according to the Output Format Rule is larger than the LRECL size, the output message is truncated. The format of the record that is written is identical to the secondary transaction generation message except the control fields, LLZ1Z2, are not included. If the data set is defined as RECFM=VB, the first 4 bytes are a halfword length, followed by halfword binary zeros, and the data. If the output message is shorter than LRECL, the remainder of the record is padded with blanks.

The SECTRX data set is not repositioned when RESTART is performed for batch transaction driver. If DISP=MOD is used, additional records will be added to the data set, and it is possible that duplicate records may occur.

If the batch transaction driver is a BMP, a secondary transaction may be routed to the IMS/VS message queue or to the SECTRX OS/VS data set as indicated in the Message Data Base. If the message is routed to the IMS/VS message queue, the batch transaction driver PSB must be based on the BUILDT9B built PSB. BUILDT9B includes the specification for the modifiable alternate PCB required.

CHECKLIST

1. Add STX operand(s) to the GENERATE statement for the Input Transaction Rule to indicate the Output Format Rule ID, the processing mode and condition for generation, and whether a new MOD name will be used.

2. If field values are to determine the conditions for secondary transaction generation, build the Audit Descriptors in the Audit Data Base and add the appropriate audit operands to the FIELD statements (i.e., AUD, FAUD). Audit Descriptor 08 is the secondary transaction operation descriptor.

3. Cause the Rules Generator to generate an Output Format Rule for each STX operand. Build the SEGMENT (TYPE=OUT) and FIELD statements that define the message layout. The GENERATE statement with OPTIONS=SEGL will generate an Output Format Rule.

4. Add segments to the Message Data Base to indicate where the secondary transaction or output message is to be routed.

5. If the transaction is a batch transaction driver, writing messages to OS/VS sequential data set SECTRX, an appropriate DD statement must be included in the JCL. An example follows:

```
EX.    //SECTRX   DD  DSNAME=SECTRANS,DISP=(NEW,KEEP),
                      DCB=(RECFM=FB,LRECL=1024,BLKSIZE=6144),
                      SPACE=(CYL,(5,1))UNIT=  ,VOL=
```

   The LRECL maximum size is 2048.

6. If the transaction is processed by the batch transaction driver, the output format rule (SEGMENT TYPE=OUT, GENERATE OPTIONS=SEGL) must be

accessible at execution time in STEPLIB or RULESDD, since it is not available in the Batch Driver Rule.

Figure 7-3 is a summary of the Secondary Transaction Checklist.

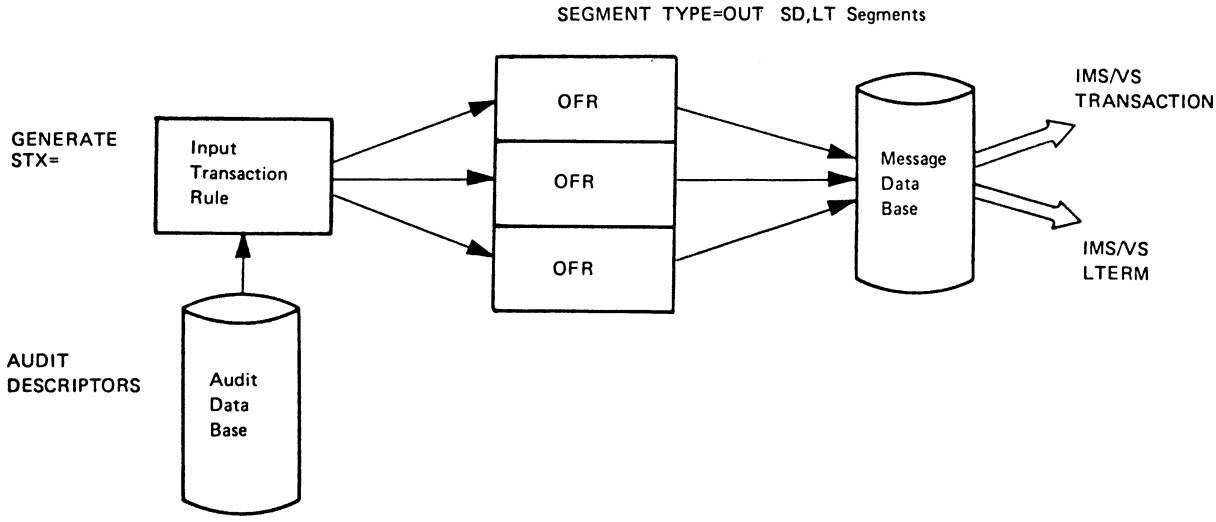


Figure 7-3. Secondary Transaction Checklist

EXAMPLE

This example uses the IMS/VS Sample Problem Parts data base and builds upon the Rules Generator source statements in the IMS Application Development Facility II Version 2 Release 2 Installation Guide. There are two conditions for generating secondary transactions and one for an alternate output message.

1. Generate the secondary transaction, SAMPTOST, whenever the field STCK (total stock) in the IV segment falls below 101. Include in the transaction the following data:

   - concatenated key of the IV segment
   - the following field data from the PA and IV segments
   - DESC
   - PRIC
   - CDAY
   - REQC
   - REQU
   - ONOR
   - STCK

2. Generate the secondary transaction, ORDRTOCT, whenever an IV segment is deleted. Include in the transaction the following data:

   - Processing mode of 3 for target segment IV
   - Concatenated key of the IV segment
   - ONOR
   - STCK

3. Generate a formatted output message to a printer LTERM (lterm name=LTERM3) whenever an error occurs during addition of an IV segment. Include the following data for display:

   - INVENTORY LOCATION -
   - Key fields for IV segment in error
   - FOR PART NUMBER -
   - Key field for PA segment
   - COULD NOT BE ADDED
   - SEGMENT ALREADY EXISTS

   A. Generate an Input Transaction Rule for IV, reflecting the requirements and conditions for secondary transaction generation. The Output Format Rule IDs are ST, IO, IE respectively.

```
*****************************************************************
*  GENERATE INPUT TRANSACTION RULE FOR IV
*****************************************************************
   GENERATE OPTIONS=INTR,SEGMENTS=IV,
                STX=(TRX,ST,0),      SECONDARY XACT FROM AUDITING STCK
                STX=(TRX,IO,OK,3),   SECONDARY XACT ON SUCCESS DLET OF IV
                STX=(MFS,IE,ER,4)    OUTPUT MESSAGE ON ADD ERROR
```

   B. Since the SAMPTOST secondary transaction will be generated according to audit criteria, the FAUD=YES and AUDIT=YES operands should be added to the FIELD statement for ID=STCK. The FAUD operand will always force an audit on the STCK field, whether or not it has been changed.

   C. Generate the three Output Format Rules, ST, IO, IE.

```
*****************************************************************
*  OUTPUT FORMAT RULE FOR SECONDARY TRANSACTION SAMPTOST (NCONV-SPEC PROC)
*****************************************************************
   SEGMENT  ID=ST,TYPE=OUT
    FIELD TEXT='SAMPTOST '            IMS/VS TRANSACTION NAME
    FIELD TEXT='0ST'                  MODE & SPECIAL PROC ID
    FIELD KWNAME=SPAKEYID             CONCATENATED KEY OF PA,IV
    FIELD ID=DESC,SEGID=PA,LEN=20     PART DESCRIPTION
    FIELD ID=PRIC,TYPE=DEC,SEGID=IV,LEN=9  UNIT PRICE
    FIELD ID=CDAY,SEGID=IV,LEN=3      STOCK DATE
    FIELD ID=REQC,TYPE=DEC,SEGID=IV,LEN=7  RQMNTS CURRENT
    FIELD ID=REQU,TYPE=DEC,SEGID=IV,LEN=7  RQMNTS UNPLANNED
    FIELD ID=ONOR,TYPE=DEC,SEGID=IV,LEN=7  ON ORDER
    FIELD ID=STOK,TYPE=DEC,SEGID=IV,LEN=7  TOTAL STOCK

*****************************************************************
*  OUTPUT FORMAT RULE FOR SECONDARY TRANSACTION ORDRTOCT (NCONV STD SEG)
*****************************************************************
   SEGMENT  ID=IO,TYPE=OUT
    FIELD TEXT='ORDRTOCT '                IMS/VS TRANSACTION NAME
    FIELD TEXT='3IV'                      MODE & SEGID (SID SEG PROC)
    FIELD KWNAME=SPAKEYID                 CONCATENATED KEY OF PA,IV
    FIELD ID=ONOR,TYPE=DEC,SEGID=IV,LEN=7  ON ORDER
    FIELD ID=STCK,TYPE=DEC,SEGID=IV,LEN=7  TOTAL STOCK

*****************************************************************
*  OUTPUT FORMAT RULE FOR IV ADD ERROR
*****************************************************************
   GENERATE OPTIONS=OMFS,SPOS=SIMAGE,PRINTER=1,ORID=IE
    SEGMENT ALREADY EXISTS

    PART NUMBER       &6KEY

    INVENTORY LOCATION - &6W  &6AREA &6INVD  &6PROJ

   &END
*****************************************************************
*  GENERATE OUTPUT FORMAT RULES FOR SECONDARY XACTS & ALTERNATE MESSAGE
*****************************************************************
   GENERATE OPTIONS=SEGL,SEGMENTS=(ST,IO)
```

D. Make entries in the Message Data Base to reflect secondary transaction and alternate output message routing. SD segments are not required for secondary transactions written to a sequential file, SECTRX, executing batch IMS.

SEGID	KEY (OUTPUT FORMAT RULE NAME)	DEFAULT DESTINATION	
SD	SAORST01	SAMPTOST	(IMS transaction name)
SD	SAORIO01	ORDRTOCT	(IMS transaction name)
SD	SAORIE01	LTERM3	(IMS LTERM name)

FAST PATH APPLICATION DESIGN CONSIDERATIONS

There are a number of functional restrictions when accessing Fast Path data bases using full-function DL/I data base calls. Some of the restrictions are specific to Fast Path and deal with what operations may be performed on different Fast Path data base and segment types, and the details of how the Fast Path calls operate, while others concern IMSADF II processing facilities for retrieving, updating, inserting and deleting (where applicable) Fast Path data base segments.

MAIN STORAGE DATA BASES - APPLICATION DESIGN

An MSDB is a root-only, in-storage data base with a single fixed-length segment type. The maximum segment length supported by IMSADF II is 6000 bytes, and the maximum key length is 70 bytes. Each MSDB is loaded in main storage at IMS/VS startup time. There are basically two types of MSDB's: Terminal-Related MSDBs and Non-Terminal-Related MSDBs.

In a Terminal-Related MSDB, each segment occurrence is "owned" by a particular IMS/VS Logical Terminal (LTERM). Any segment may be retrieved by any terminal, but a segment can only be updated by a transaction originating from the logical terminal which owns the segment. The segment key of the Terminal-Related MSDB is the Logical Terminal (LTERM) name of the terminal which owns (and can update) that segment occurrence. This "terminal-related key" is not physically part of the segment, but is available from the key-feedback area of the data base PCB.

The TYPE=DBEK parameter on the SEGMENT statement for the Rules Generator can be used to make the key value appear to be contained in the segment.

A Terminal-Related MSDB may be "Fixed" (segments can be retrieved, or replaced (by the owning LTERM), but cannot be inserted or deleted), or "Dynamic" (segments may be retrieved by any LTERM, or replaced, inserted, or deleted by the owning LTERM).

In a Non-Terminal-Related MSDB, there is no special relationship between segment occurrences and IMS/VS Logical Terminals for segment updates. Non-Terminal-Related MSDB segments may be retrieved or replaced, but not inserted or deleted. There are two sub-types of Non-Terminal-Related MSDB. In a Non-Terminal-Related MSDB with terminal-related keys, the segment key is the name of an IMS/VS LTERM, and this key is not physically part of the segment, but is available from the key-feedback area of the data base PCB. In a Non-Terminal-Related MSDB without terminal-related keys, the segment key may be any value, and is physically part of the segment. Non-Terminal-Related MSDBs can be used instead of a small DASD root-only data base, for application control information or for tables (e.g. rate tables, edit tables and table lookup, schedules).

MSDB Data Base Access Characteristics

The following is a summary of the types of DB calls allowed, and access characteristics, pertaining to the different types of MSDBs. DL/I calls or facilities not mentioned will normally work the same way for MSDBs as for non-Fast Path data bases.

- MSDB - Terminal-Related Fixed

  Any segment can be retrieved. Only the segment related to a terminal can be updated. No segment can be deleted or inserted. The segment key is not contained in the segment data.

- MSDB - Terminal-Related Dynamic

  Any segment can be retrieved. Only the segment related to a terminal can be updated, inserted, or deleted. The segment key is not contained of the segment data.

- MSDB - Non-Related With Terminal-Related Key

Any segment can be retrieved or updated by any terminal. No segment can be inserted or deleted. The segment key is not contained in the segment data.

- MSDB - Non-Related Without Terminal-Related Key

  Any segment can be retrieved or updated. No segment can be inserted or deleted. The key is part of the segment data.

The following MSDB design restrictions/considerations have been divided into two groups: for MSDBs with Terminal-Related Keys, and for MSDBs without Terminal-Related Keys.

MSDB WITH TERMINAL-RELATED KEYS

- Results of MSDB REPL/ISRT/DLET Calls: The results of a DL/I update call are not visible to the program until program termination. For example, an ISRT call, followed by a GHU call for the ISRTed segment will result in a 'GE' call status code for the GHU call.

- Accessing Segments Using *MYLTERM

  The IMSADF II Fast Path Data Base support facility, using the Fast Path special segment name '*MYLTERM' in the SSA, allows access to the MSDB record related to the source logical terminal by entering a single asterisk ('*') as a selection key when the LTERM name is not known. For applications where security is important it is recommended to use the Audit Rules to force the user to enter the correct LTERM name.

- Updating of Related Segments

  For Terminal-Related MSDBs, only the segment owned by a terminal can be updated. However, IMSADF II cannot tell whether a segment is related to a terminal, and will accept a segment transaction mode of 5, after checking the Sign-on DB profile segment, even if the segment is not related to the Logical Terminal entering the transaction. An attempt to update an unowned segment will result in an 'AM' status code from DL/I. In order to avoid trying to update an unowned segment, it is recommended to use the Audit Rules to force the application user to enter a LTERM name of '*' to access only the related segment. Only read-only applications should be allowed to enter LTERM names.

- Secondary Key Selection and Key Display Support

  For MSDBs with Terminal Related Keys the (LTERM name) is not part of the segment data which means that ordinarily the key data cannot be displayed and Secondary Key Selection cannot be used because the first eight bytes of the segment will be used as the key. However, the support facility allows display of the key data and use of Secondary Key Selection if the application designer sets up the Static Rules as follows:

  1. The SEGMENT statement must have the TYPE parameter defined with the value DBEK (TYPE=DBEK). This indicates that the key is not really part of the segment but that IMSADF II should build a segment area including the key.

  2. The first FIELD statement must be for the LTERM data, starting at position 1 with a length of 8, and must be defined with "KEY=YES" to indicate that this field is indeed the segment key.

     The first actual field of the segment starts at position 9.

MSDB WITHOUT TERMINAL RELATED KEYS

- Results of MSDB Calls to Changed Segments: The results of a DL/I call are not visible to the program until after transaction termination (IMS/VS sync. point). For example, a REPL call, followed by a GU call for the replaced segment, will get the segment data as it was when the transaction started, not as it was after the REPL call.

DATA ENTRY DATA BASES - APPLICATION DESIGN

A Data Entry Data Base (DEDB) contains a root segment and as many as 127 dependent segment types. One of these can be a sequential dependent; the other 126 are direct dependents. Sequential dependent segments are stored in chronological order, regardless of the root that they are dependent on. Direct dependent segments are stored hierarchically. DL/I calls to DEDBs can include as many SSAs as there are levels in the hierarchy (a maximum of 15). Root and direct dependent segment can be deleted, inserted, updated, and retrieved. Sequential dependent segments may only be inserted and retrieved. Delete or replace attempts will result in an 'AM' status code from DL/I.

DEDB design restrictions/considerations have been divided into two groups (DEDB - All Segment types, and DEDB - Sequential Dependent Segment Type. Further information regarding restrictions on Fast Path data bases and IMSADF II processing can be found in some of the manuals mentioned in the Preface.

DEDB CONSIDERATIONS - ALL SEGMENT TYPES: The following design considerations apply to DEDB root and direct dependent segments, as well as to Sequential Dependent segments.

- DEDB Segment Format - Variable Length

  All segments in a DEDB are variable-length segments. Thus the application code (eg. audit rules) must maintain the segment length field, and be prepared to handle short segments and missing fields if they occur. Path calls should not be used.

DEDB CONSIDERATIONS - SEQUENTIAL DEPENDENTS: The following design considerations apply to DEDB Sequential Dependent segments only.

- Sequential Dependent Segment Replace/Delete

  Sequential dependent segments in a DEDB can only be retrieved (GU, GHU, GN(P), GHN(P)) and inserted (ISRT), and cannot be updated or deleted. However, IMSADF II will accept a segment transaction mode of 5 (update), if the segment is allowed to be inserted, after checking the Sign-on DB profile segment. In order to avoid updating, it is recommended to use the Audit Rules to check for a transaction mode of 5 and terminate by sending a message.

- Sequential Dependent Segment Insert/Retrieve

  Sequential Dependent segments in a DEDB are inserted in last-in first-out (LIFO) order, and do not have a sequence field. In order to Insert/Retrieve Sequential Dependent Segments with IMSADF II the following must be considered:

  - Since IMSADF II expects all segments to have a key field (for example, non-keyed segments are not supported by IMSADF II) the Static Rules require that a key field be specified for the sequential segment in the SEGMENT and FIELD statements. Therefore the application designer must supply a key field by using one of the existing fields or by creating a new one.

  - Insertion of Sequential Dependent Segments by normal IMSADF II processing will result in an attempt by IMSADF II to retrieve the segment to be inserted which will in turn result in:

    1. Performance impact if many Sequential Dependent Segments have already been inserted.

    2. Insertion not allowed if the key already exists.

       Recommendation: use the Audit language or Special Processing (SPRs) to insert Sequential Dependent Segments.

- Secondary Key Selection of Sequential Dependents

  Secondary Key Selection should not be used for Sequential Dependent segments, since these segment types do not have a sequence field

(key field). However, if it is used, the following points should be remembered:

1. The keys of the Sequential Dependent segments will be displayed in LIFO sequence (based on order of insertion). If the keys are in descending order only the number of keys specified in the SEGMENT SKSEGS statement will be displayed. The reason for this is that IMSADF II, when the next set of keys is requested, will issue a qualified GNP to access the next segment which must have a key value greater than the last segment retrieved in the previous set of displayed keys.

2. A 'GNP' call will be issued to retrieve every Sequential Dependent segment under that root thus impacting the response time.

   Recommendation: do not allow secondary key selection for Sequential Dependent segment types.

CHAPTER 8. IMSADF II UNDER CICS/OS/VS

DEFINING DFHPPT ENTRIES

The CICS DFHPPT macro defines load modules for CICS/OS/VS. A sequential data set, CICSPPT, is created by the Rules Generator which contains DFHPPT definition statements for all the load modules defined during a Rules Generator run.

The load modules created for an IMSADF II application include:

- static rules
- transaction drivers
- mapsets

Static rules and transaction drivers are described elsewhere in this manual. Mapsets are described in the following section.

MAPSETS

A mapset is a CICS control block, containing one or more maps, used by CICS' Basic Mapping Support (BMS) to describe the layout of a display screen. A mapset usually contains maps for more than one screen, and for each screen, there is a map for each screen size requested. The Rules Generator produces a partitioned data set, CICSMAPS, with each member containing the BMS screen source defining a different mapset. Each member of the PDS is input to the CICS BMS Utility to create the mapset load module. The mapsets created by IMSADF II do not use mapset suffices. You may realize better performance if you specify NODDS for the BMS operand in your CICS System Initialization Table (DFHSIT).

Nonconversational

A mapset created by the Rules Generator for a nonconversational transaction contains the map(s) for the data display screen, one for each different DEVTYPE defined. The MODNAME operand of the GENERATE statement defines the mapset name for a transaction.

Conversational

One of the mapsets created by the Rules Generator for a conversational transaction contains the map(s) for the Sign-on Screen, one map for each screen size requested. The naming convention for this mapset is:

ssssAa

where,

ssss
is the application system ID (SYSID)

a
is the transaction trailer (TRXTRLR)

Another mapset created by the Rules Generator for a conversational transaction contains the maps unique to a particular IMSADF II transaction:

- primary key selection screen map(s)
- segment display screen map(s)

The naming convention for this mapset is:

ssttCm

where,

ss
is the first two characters of the application system ID (SYSID)

tt
is the IMSADF II transaction ID

m
is the BMS (MFS) trailer specified at installation time

A mapset created during installation contains the 'base' screens:

- primary option menu screen map(s)
- secondary option menu screen map(s)
- secondary key selection screen map(s)
- project message sending screen map(s)
- project message display screen map(s)
- user message sending screen map(s)
- user message display screen map(s)
- error message screen map(s)

The naming convention for this mapset is:

xxxxBm

where,

xxxx
is the four-character ADFID

or the alternate language SYSID (NLS only)

m
is the BMS (MFS) transaction trailer (MFSTRLR)

SCREEN SIZE CONSIDERATIONS

The mapset for an IMSADF II transaction contains a map definition for each screen size as specified by the DEVTYPE operand of the GENERATE statement. When IMSADF II issues a CICS SEND (or a CICS RECEIVE) command, a different map is referenced depending on the screen size of the terminal. The screen size of the terminal can be affected by CICS DFHPCT and DFHTCT entries for the transaction and terminal used to start the IMSADF II session.

- The DFHPCT entry may be defined with the SCRNSIZE operand as either DEFAULT or ALTERNATE.

- The DFHTCT entry may be defined with the default (DEFSCRN) and alternate (ALTSCRN) screen size.

The Rules Generator specification, the DFHPCT specification, and the DFHTCT specification must all be consistent to work properly.

EXAMPLE:

Assume the following specifications for the Rules Generator, DFHPCT and DFHTCT respectively:

```
GENERATE DEVTYPE=(3,4),...

DFHPCT SCRNSIZE=ALTERNATE,...

DFHTCT DEFSCRN=(24,80),ALTSCRN=(43,80),...
```

This would cause IMSADF II to use the map for a screen size of 43 by 80. Attempting to run this transaction from a device with fewer than 43 lines would result in a CICS ABM0 application ABEND because IMSADF II would attempt to send a map that does not exist in the mapset. A CICS ABM0 application ABEND would also result if the DFHPCT specification were changed to:

```
DFHPCT SCRNSIZE=DEFAULT,...
```

DEFINING THE PSBS

A Program Specification Block (PSB) is an IMS/VS control block which defines a application programs view of DL/I data bases, and PSBGEN is the name of the IMS/VS procedure used to generate a PSB. If your IMSADF II application requires access to a DL/I data base, then a PSB must be generated. See "Defining the PSBs for IMSADF II Applications" in Chapter 7, "IMSADF II under IMS/VS." The PSBGEN is exactly the same for subsequent execution of either IMS/VS or CICS/OS/VS. (CICS code appropriately ignores the alternate PCB definitions included in Assembler macros BUILDT07 or BUILDT08.)

Normal CICS/OS/VS resource definition of DL/I data bases is necessary. For additional information refer to the DFHDLPSB and DFHDLDBD macros in the CICS/VS Resource Definition Guide.

VSAM FILES

You may access VSAM files using IMSADF II in a CICS/OS/VS environment. The DFHFCT macro is used to define VSAM files to CICS/OS/VS. VSAM files are accessed in the same way as root-only DL/I data bases. IMSADF II simulates the DL/I interface in the CICS/OS/VS environment, accessing VSAM files using the CICS file control commands.

The TYPE operand of the SEGMENT statement is used in the Rules Generator to define a VSAM file:

```
TYPE=   KSDS  for Key Sequenced Data set
    =   ESDS  for Entry Sequenced Data Set
    =   RRDS  for Relative Record Data Set
    =   RREK  for Relative Record Data Set with an External Key

PARENT= 0

NAME=   the DATASET name as used in the DFHFCT entry
```

See also "VSAM Calls" on page 3-19 in Chapter 3, "Audit Language."

Note: To be used with IMSADF II, a VSAM data set must contain at least one record.

The four types of VSAM files which IMSADF II supports are:

Key Sequenced Data Set (KSDS)

Access to a KSDS file can only be made via the primary key. Records can be retrieved, added, updated, and deleted using normal IMSADF II functions.

Entry Sequenced Data Set (ESDS)

ESDS files are only supported for output. New records are simply added at the end of the data set. Attempts to retrieve, update, or delete an ESDS record will result in an appropriate error message from IMSADF II.

Relative Record Data Set (RRDS)

The data records in a RRDS file are located based on their relative record number. Records can be retrieved, added, updated, and deleted using normal IMSADF II functions.

Relative Record Data Set with an External Key (RREK)

Records in a RREK file are accessed using the relative record number as the key and can be retrieved, added, updated, and deleted. For a RREK file, you must define an additional field in the IMSADF II view of the record, used to hold the relative record number for IMSADF II processing. This must be defined as a binary fullword and must be the first field in the record.

DEFINING BATCH TRANSACTION DRIVER JCL

The CICS/OS/VS startup jobstream should include the following DD statements as appropriate, to allow execution of an IMSADF II batch transaction driver. These data sets must be defined to CICS/OS/VS as Extrapartition Transient Data data sets via the CICS DFHDCT macro. The DDNAMES are user defined in the DFHDCT; the names shown here are the **recommended** names to be used. For additional information, see the IMS Application Development Facility II Version 2 Release 2 Installation Guide.

DDNAME	DESTID	
PRINTER	PRIN	Defines the hard copy output from the batch run.
RSTRTIN	RSTR	Defines the sequential input data set that is to contain the restart (RSTRT) transaction if a batch job is being restarted. DD DUMMY can be specified if the job is a normal run.
TRANSIN	ADFI	Defines the sequential input data set that is to contain the transaction messages to be processed.
TRANSOUT	ADFO	(Optional) Defines the sequential output data set that is to contain the unprocessed transaction messages if STOP has been entered by the OS/VS System Operator. If omitted, the WTOR message to the OS/VS System Operator is not issued.
ERRMSG	ERRM	(Optional) Defines the hard copy error message data set. All data base transactions that are in error and their associated error messages may be logged here.
ERRTRX	ERRT	(Optional) Defines a data set to which error transactions may be written.
SECTRX	SECT	(Optional) Defines a sequential data set to receive secondary transaction records when executing a batch transaction driver. See "Secondary Transaction/Output Message Routing." for additional information.

EXAMPLE:

```
//BDCICS   JOB   ACCNT,NAME,MSGLEVEL=1
//CICS     EXEC  DFHSIP
    .
    .
    .
//PRINTER  DD    SYSOUT=A
//RSTRTIN  DD    DUMMY
//TRANSIN  DD    DSN=USER.TRANSIN(MEMBER1),DISP=SHR
//*
//*      DATA BASE DD STATEMENTS
//*
//DB1      DD    DSN=USER.DB1,DISP=OLD
//DB2      DD    DSN=USER.DB2,DISP=OLD
```

MULTIREGION OPERATION (MRO)

Multiregion Operation (MRO) is a facility of CICS/OS/VS that allows multiple regions within the same system to communicate with each other. CICS MRO has three main functions:

- **Transaction routing,** which allows operators of terminals in one CICS region to run transactions in any connected CICS region. There is also a CICS-provided routing transaction.
- **Function shipping,** which allows command-level application programs (or programs written for the DL/I interface) to address resources in any connected CICS region.

- **Distributed transaction processing,** which allows an application to be written as complementary transactions, each executing in an MRO connected region.

CICS MRO functions are implemented and controlled via CICS and do not require any changes to IMSADF II.

SECONDARY TRANSACTION/OUTPUT MESSAGE ROUTING

When indicated in the Input Transaction Rule, the transaction driver sends a message defined by the Output Format Rule (OFR) to a destination defined in the Message Data Base. The destination may be a program (secondary transaction) or one or more terminals (message route). If the ultimate destination is a program, the Message Data Base contains the name of a CICS Intrapartition Transient Data queue for the program. The DFHDCT entry defines the CICS transaction code (TRANSID) of the target application program to receive the message. The secondary transaction can be a non-Facility application program or an IMSADF II nonconversational transaction driver.

To generate a secondary transaction or alternate output message, the conditions for generation, the format of the output, and the destination must be specified using a combination of Rules Generator statements and Message Data Base and Audit Data Base entries. The Input Transaction Rule lists the secondary transactions that may be generated and the conditions under which they will be generated. The conditions for generation include the processing mode (0 to 6) and the processing completion (OKAY or ERROR). Field auditing also can be used to determine secondary transaction generation. When a field meets a specified set of audit criteria, the secondary transaction operation descriptor, 08, can be specified to indicate which secondary transaction is to be generated. (Refer to Chapter 5, "Audit Logic Processing.") If this Input Transaction Rule is for Special Processing, a SPA flag, SPASECTX, is available to the Special Processing Routine to tell the Transaction Driver whether or not to invoke secondary transaction generation and for what conditions the Driver should check. To incorporate secondary transaction processing the Input Transaction Rule, the STX operand(s) should be specified on the GENERATE statement.

The format of the STX operand is:

```
STX=( MFS ,xx [,OK][,ER] ,mode)
      TRX
```

where,

MFS — indicates that the message is to be routed to the terminal(s) defined in the Message Data Base.

TRX — indicates that the message is to be written to the CICS Transient Data destination defined in the Message Data Base.

xx — specifies the Output Format Rule ID. The OFR defines the layout of the message. The xx is expanded by the Rules Generator to become an eight-character OFR name. The expansion is ssORxx01, where

```
ss = first two characters of the application System ID
OR = constant 'OR'
xx = Output Format Rule ID
01 = constant '01'
```

OK — indicates that the message will be generated if the processing (data base update) was successful.

ER — indicates that the message will be generated if the processing (data base update) was unsuccessful.

mode — indicates the current processing mode (1 to 6) for which this secondary transaction should be generated. A value of 0 indicates any mode 1 to 6.

If field auditing is to determine whether the secondary transaction is generated, neither OK nor ER need be specified.

Refer to Figure 7-2 on page 7-11 for a summary of the logic among Driver processing, STX operand values, and generating secondary transactions.

OUTPUT FORMAT

The format of the output message is defined by the Output Format Rule. There must be an Output Format Rule for each STX operand specified. The Output Format Rule itself is generated by SEGMENT and FIELD statements that define the layout of the message, which is a maximum of 1600 bytes. The operands TYPE=OUT and ID=XX, (where XX matches the XX of the STX operand) are specified on the SEGMENT statement. These statements will define the layout of the secondary transaction or alternate output message to be generated. Therefore, it must match the BMS mapset name used for output to a terminal. If this is a secondary transaction to be processed by the Nonconversational Driver, the Output Format Rule must have the first fields defined as follows:

- Eight-character transaction name
- blank character (x'40')
- mode
- segment ID or Special Processing ID to be processed

The contents of the message comes from three sources - field data from segments currently processed, text data from the Output Format Rule, and keyword data from the SPA or segment handler communication area. Refer to the SEGMENT statement, operand TYPE=OUT, for more detail.

MESSAGE ROUTING

To determine the destination name (terminal or Transient Data queue), IMSADF II accesses the Message Data Base as described in "Message Data Base" on page 4-12. The key is the Output Format Rule name. A message may be routed to up to eight specified terminals.

A batch transaction driver writes the message to an Extrapartition Transient Data data set for which the destination name is SECT. The format of the record that is written is identical to the secondary transaction generation message except the control fields, LLZ1Z2, are not included. See "Defining Batch Transaction Driver JCL" for Batch Driver Extrapartition Transient Data destinations.

CHECKLIST

1. Add STX operand(s) to the GENERATE statement for the Input Transaction Rule to indicate the Output Format Rule ID, and the processing mode and condition for generation.

2. If field values are to determine the conditions for secondary transaction generation, build the Audit Descriptors in the Audit Data Base and add the appropriate audit operands to the FIELD statements (i.e., AUD,FAUD). Audit Descriptor 08 is the secondary transaction operation descriptor.

3. Generate an Output Format Rule for each STX operand. Build the SEGMENT (TYPE=OUT) and FIELD statements that define the message layout. The GENERATE statement with OPTIONS=SEGL will generate an Output Format Rule.

4. Add segments to the Message Data Base to indicate where the secondary transaction or output message is to be routed.

5. If transaction is a batch transaction driver, writing messages to CICS Extrapartition Transient Data destination SECT, an appropriate DD statement must be included in the CICS startup JCL. The recommended DDNAME is SECTRX. An example follows:

```
//SECTRX    DD  DSNAME=SECTRANS,DISP=(NEW,KEEP),
                DCB=(RECFM=FB,LRECL=1024,BLKSIZE=6144),
```

SPACE=(CYL,(5,1)),UNIT=SYSDA

6. If the transaction is processed by the batch transaction driver, the Output Format Rule must be accessible at execution time in DFHRPL since it is not available in the Batch Driver Rule.

APPENDIX A. SUMMARY OF RULES GENERATOR OPERAND USAGE

The Rules Generator control statements contain a set of operands that define rule and screen naming conventions, screen headings and field literals, field positioning and mode, key selection and transaction parameters, segment search arguments, field storage attributes, and field audit status. The following figures show the Rules Generator operand usage for each rule and screen for each of the transaction driver types, Conversational, Nonconversational, and Batch. The GENERATE statement OPTIONS operand values used to create the rules and screens of this figure are shown immediately below the rule or screen heading. The Secondary Key Selection screen is shown as a table entry, even though a predefined screen is being used, because the actual screen image data is generated by the Rules Generator as part of the Input Transaction Rule. The SEGMENTS operand is used to select the segments for segment related rules generation. The SYSID operand is used as four characters in defining rule names. Only the first two characters of this operand are used to define the screen names. The operands that control the Rules Generator operation (DECKS, LINKREQ, ASMLIST, MEMBER, LOPT, and LINEC, etc.) do not influence the content of the rules generation and, therefore, are not shown in this table.

The one-character entries in the rules generation table indicate the following:

- R required operand
- D default value available for required operand
- X optional operand

	RULES									SCREENS						
	Segment/Table Layout	Segment/Table Handler	Input Transaction	Standard Link Edit	Primary Option Menu	Secondary Option Menu	Text Input Transaction	Conv SP Link Edit	Preload	Sign On	Primary Key Select	Secondary Key Select	Seg Display Auto	Seg Display Row Column	Seg Display SIMAGE	Text Utility
	SEGL	SEGH	INTR	STLE	POM	SOM	TEXT	SPLE	PREL	SIGN	KEYD		SEGD	SEGD	SEGD	TEXT
ADFID				D				D		D						
AGROUP			X													
DBID	D	D	D				D									
ENVIR										D	D		D	D	D	
KANAME			X													
LRULE			X													
MAXKEY													X	X	X	
PCBNO			D				D									
PGROUP			R				R								X	
POMENU					R											
SFORMAT											D		D	D		
SHEADING										X	X		X	X		X
SIGNON										X						
SOIMAGE										X						
SOMTX						R										
STRAILER										D	D		D	D	D	D
SYSID	R	R	R		R	R	R	R	R	R	R		R	R	R	R
USRLANG										D	D		D	D	D	D

Figure A-1. Summary: SYSTEM Statement (Conversational)

	RULES									SCREENS						
	Segment/Table Layout	Segment/Table Handler	Input Transaction	Standard Link Edit	Primary Option Menu	Secondary Option Menu	Text Input Transaction	Conv SP Link Edit	Preload	Sign On	Primary Key Select	Secondary Key Select	Seg Display Auto	Seg Display Row Column	Seg Display SIMAGE	Text Utility
	SEGL	SEGH	INTR	STLE	POM	SOM	TEXT	SPLE	PREL	SIGN	KEYD		SEGD	SEGD	SEGD	TEXT
BYTES	R															
DBID	D	D	D													
DCFIELD	D															
DISPLAY			D										D	D		
ID	R	R														
KASCEND			D				D									
KEYNAME		R														
LENGTH	R															
MODE			D													
NAME		R														
NEWID	X	X														
OVERRIDE	X															

Figure A-2 (Part 1 of 2). Summary: SEGMENT Statement (Conversational)

	RULES									SCREENS						
	Segment/Table Layout	Segment/Table Handler	Input Transaction	Standard Link Edit	Primary Option Menu	Secondary Option Menu	Text Input Transaction	Conv SP Link Edit	Preload	Sign On	Primary Key Select	Secondary Key Select	Seg Display Auto	Seg Display Row Column	Seg Display SIMAGE	Text Utility
	SEGL	SEGH	INTR	STLE	POM	SOM	TEXT	SPLE	PREL	SIGN	KEYD		SEGD	SEGD	SEGD	TEXT
PARENT		R	R				R									
PCBNO			D				D									
SKLEFT			D				D					D				
SKRIGHT			D				D					D				
SKSEGS			D				D					D				
SQLDIST		X														
SQLIND	X															
SQLISRT		X														
SQLNAME		R														
SQLUPD		X														
TRAILER	D		D				D									
TYPE	D		D				D									

Figure A-2 (Part 2 of 2). Summary: SEGMENT Statement (Conversational)

	RULES									SCREENS						
	Segment/Table Layout	Segment/Table Handler	Input Transaction	Standard Link Edit	Primary Option Menu	Secondary Option Menu	Text Input Transaction	Conv SP Link Edit	Preload	Sign On	Primary Key Select	Secondary Key Select	Seg Display Auto	Seg Display Row Column	Seg Display SIMAGE	Text Utility
	SEGL	SEGH	INTR	STLE	POM	SOM	TEXT	SPLE	PREL	SIGN	KEYD		SEGD	SEGD	SEGD	TEXT
AFA			X													
ASTATUS	X		X													
AUDIT	X															
BITOFF	D		D													
BYTES	R	R	R				R				R	R	R	R	R	R
CAUDIT	X															
COFIELD			X				X				X	X				
COLUMN			D				D					D				
COMPARE		X														
DECIMAL	D		D				D									
DISPLAY			D										D	D		
DMODE			D													
FAUDIT			X													
FLDOL			X													
ID	R															
KAUDIT			D				D									

Figure A-3 (Part 1 of 3). Summary: FIELD Statement (Conversational)

	RULES									SCREENS						
	Segment/Table Layout	Segment/Table Handler	Input Transaction	Standard Link Edit	Primary Option Menu	Secondary Option Menu	Text Input Transaction	Conv SP Link Edit	Preload	Sign On	Primary Key Select	Secondary Key Select	Seg Display Auto	Seg Display Column Row	Seg Display SIMAGE	Text Utility
	SEGL	SEGH	INTR	STLE	POM	SOM	TEXT	SPLE	PREL	SIGN	KEYD		SEGD	SEGD	SEGD	TEXT
KDISPLAY			D				D				D	D	D	D	D	
KEY		R	X								D	D				
KEYBLD			X				X				D	D				
KSELECT			X				X									
LENGTH	R	R	R				R				R	R	R	R	R	R
MODE			D										D	D		
MSG	X															
NAME	R															
OFFSET	X															
PAUDIT			X													
POSITION	D															
REDEFINE	X															
RELATED			X				X					X				
REQUIRED			X													
SCOL														R		
SDECIMAL			D													

Figure A-3 (Part 2 of 3). Summary: FIELD Statement (Conversational)

	RULES									SCREENS						
	Segment/Table Layout	Segment/Table Handler	Input Transaction	Standard Link Edit	Primary Option Menu	Secondary Option Menu	Text Input Transaction	Conv SP Link Edit	Preload	Sign On	Primary Key Select	Secondary Key Select	Seg Display Auto	Seg Display Row Column	Seg Display SIMAGE	Text Utility
	SEGL	SEGH	INTR	STLE	POM	SOM	TEXT	SPLE	PREL	SIGN	KEYD		SEGD	SEGD	SEGD	TEXT
SIGN	D		D				D	D								
SLENGTH			D										D	D	D	
SNAME											X	X	X	X		
SQLISRT		X														
SQLNAME		X														
SQLNULL	X															
SQLORD		X														
SQLUPD		X														
SROW														R		
START	D															
TYPE	D		D				D	D								
VALUE		X														
VCOL			X													
VMODE			X													
VROW			X													

Figure A-3 (Part 3 of 3). Summary: FIELD Statement (Conversational)

	RULES									SCREENS						
	Segment/Table Layout	Segment/Table 4andler	Input Transaction	Standard Link Edit	Primary Option Menu	Secondary Option Menu	Text Input Transaction	Conv SP Link Edit	Preload	Sign On	Primary Key Select	Secondary Key Select	Seg Display Auto	Seg Display Row Column	Seg Display SIMAGE	Text Utility
	SEGL	SEGH	INTR	STLE	POM	SOM	TEXT	SPLE	PREL	SIGN	KEYD		SEGD	SEGD	SEGD	TEXT
ADDMODE			D													
AEXIT				X				X								
AGROUP			D													
ALANG				X				X								
ALIAS	X	X														
ASRTABLE									X							
BYPASS			X													
COMMLEN			X				X									
CURSOR													X	X	X	
DAMSG			X													
DASRTAB									X							
DATACOMP			X													
DBPATH			X				R				X		X	X	X	X
DELETE						X			X							
DEVCHRS										X	X		X	X	X	
DEVNAME										X	X		X	X	X	X
DEVTYPE										X	X		X	X	X	X

Figure A-4 (Part 1 of 4). Summary: GENERATE Statement (Conversational)

	RULES									SCREENS						
	Segment/Table Layout	Segment/Table Handler	Input Transaction	Standard Link Edit	Primary Option Menu	Secondary Option Menu	Text Input Transaction	Conv SP Link Edit	Preload	Sign On	Primary Key Select	Secondary Key Select	Seg Display Auto	Seg Display Row Column	Seg Display SIMAGE	Text Utility
	SEGL	SEGH	INTR	STLE	POM	SOM	TEXT	SPLE	PREL	SIGN	KEYD		SEGD	SEGD	SEGD	TEXT
DEXIT				X				X								
DISPNAME			X										X	X	X	
DKEY													X	X	X	
DLANG				X				X								
DLET			X													
DLIEXIT			X													
DTRAN													X	X	X	
DTWINC			X													
DYNLOAD				X				X								
IMAGE															X	
ISRT			X													
ITTABLE									X							
KANAME			X													
KEYSL			X			D										
LANGUAGE			D	D				D								
LRULE			X													
MAPTABLE									X							

Figure A-4 (Part 2 of 4). Summary: GENERATE Statement (Conversational)

	RULES									SCREENS						
	Segment/Table Layout	Segment/Table Handler	Input Transaction	Standard Link Edit	Primary Option Menu	Secondary Option Menu	Text Input Transaction	Conv SP Link Edit	Preload	Sign On	Primary Key Select	Secondary Key Select	Seg Display Auto	Seg Display Row Column	Seg Display SIMAGE	Text Utility
	SEGL	SEGH	INTR	STLE	POM	SOM	TEXT	SPLE	PREL	SIGN	KEYD		SEGD	SEGD	SEGD	TEXT
MAXKEY													X	X	X	
MSGDB		X														
MSGNF		X														
OFRTABLE									X							
OPTIONS	X	X	X	X	X	X	X	X	X	X	X		X	X	X	X
PFKDATA													X	X	X	
PFKLIT													X	X	X	
PFKNUMB													X	X	X	
PGMID				R				R								
PGROUP			R				R									
POMENU					X											
PREPOM									X							
PRESOM									X							
PRINTER									X						X	
SEGMENTS	X	X														
SFORMAT											D		D	D		
SHEADING										X	X		X	X		X

Figure A-4 (Part 3 of 4). Summary: GENERATE Statement (Conversational)

	RULES									SCREENS						
	Segment/Table Layout	Segment/Table Handler	Input Transaction	Standard Link Edit	Primary Option Menu	Secondary Option Menu	Text Input Transaction	Conv SP Link Edit	Preload	Sign On	Primary Key Select	Secondary Key Select	Seg Display Auto	Seg Display Row Column	Seg Display SIMAGE	Text Utility
	SEGL	SEGH	INTR	STLE	POM	SOM	TEXT	SPLE	PREL	SIGN	KEYD		SEGD	SEGD	SEGD	TEXT
SHTABLE									X							
SIGNON										X						
SLRTABLE									X							
SOIMAGE										X						
SOMEND						X										
SOMTX						R										
SPECIAL			X													
SPOS													D	X	X	
SPTABLE									X							
STX			X													
TPITTABL									X							
TRXID			R													
TRXNAME						X					X	X	X	X		
TSEGS			X													
TWINS			X													
UPDATE						X			X							
USRLANG										D	D		D	D	D	D
VACANT			X													

Figure A-4 (Part 4 of 4). Summary: GENERATE Statement (Conversational)

RULES SCREENS

	Segment Layout	Segment Handler	Input Transaction	Non Conv Link Edit	TP SP Link Edit	Seg Display Image
	SEGL	SEGH	TPIT	NCLE	TPLE	SEGD
ADFID				D	D	
AGROUP			X			
DBID	D	D	D			
ENVIR						D
KANAME			X			
LRULE			X			
PCBNO			D			
STRAILER						D
SYSID	R	R	R		R	R
USRLANG						D

Figure A-5. Summary: SYSTEM Statement (Nonconversational)

RULES SCREENS

	Segment Layout	Segment Handler	Input Transaction	Non Conv Link Edit	TP SP Link Edit	Seg Display Simage
	SEGL	SEGH	TPIT	NCLE	TPLE	SEGD
BYTES	R					
DBID	D	D	D			
DISPLAY			D			
ID	R	R				
KASCEND			D			
KEYNAME		R				
LENGTH	R					
MODE			D			
NAME		R				
NEWID	X	X				
OVERRIDE	X					
PARENT		R	R			
PCBNO			D			
SQLDIST		X				
SQLIND	X					
SQLISRT		X				
SQLNAME		R				
SQLUPD		X				
TRAILER	D		D			
TYPE	D		D			

Figure A-6. Summary: SEGMENT Statement (Nonconversational)

RULES SCREENS

	Segment Layout (SEGL)	Segment Handler (SEGH)	Input Transaction (TPIT)	Non Conv Link Edit (NCLE)	TP SP Link Edit (TPLE)	Seg Display Image (SEGD)
AFA			X			
ASTATUS	X		X			
AUDIT	X					
BITOFF	D		D			
BYTES	R	R	R			R
CAUDIT	X					
COFIELD			X			
COMPARE		X				
DECIMAL	D		D			
DISPLAY			D			
FAUDIT			X			
FLDOL			X			
FLDPOS			X			
ID	R					
ILENGTH			D			
ITYPE			D			
KAUDIT			D			
KEY		R	X			
KEYBLD			X			
LENGTH	R	R	R			R
MODE			X			

Figure A-7 (Part 1 of 2). Summary: FIELD Statement (Nonconversational)

RULES SCREENS

	Segment Layout SEGL	Segment Handler SEGH	Input Transaction TPIT	Non Conv Link Edit NCLE	TP SP Link Edit TPLE	Seg Display Image SEGD
MSG	X					
NAME	R					
OFFSET	X					
PAUDIT			X			
POSITION	D					
REDEFINE	X					
REQUIRED			X			
SDECIMAL			D			
SIGN	D		D			
SLENGTH			D			D
SQLISRT		X				
SQLNAME		X				
SQLNULL	X					
SQLORD		X				
SQLUPD		X				
START	D					
TYPE	D		D			
VALUE		X				
VCOL			X			
VMODE			X			
VROW			X			

Figure A-7 (Part 2 of 2). Summary: FIELD Statement (Nonconversational)

RULES SCREENS

	Segment Layout SEGL	Segment Header SEGH	Input Transaction TPIT	Non Conv Link Edit NCLE	TP SP Link Edit TPLE	Seg Display Image SEGD
AEXIT				X	X	
AGROUP			D			
ALANG				X	X	
ALIAS	X	X				
CURSOR						X
DAMSG			X			
DBPATH			X			X
DEVCHRS						X
DEVNAME						X
DEVTYPE						X
DEXIT				X	X	
DLANG				X	X	
DLET			X			
IMAGE						X
ISRT			X			
KANAME			X			
LANGUAGE			D	D	D	
LRULE			X		X	

Figure A-8 (Part 1 of 2). Summary: GENERATE Statement (Nonconversational)

RULES SCREENS

	Segment Layout (SEGL)	Segment Handler (SEGH)	Input Transaction (TPIT)	Non Conv Link Edit (NCLE)	TP SP Link Edit (TPLE)	Seg Display Simage (SEGD)
MAPTABLE					X	
MODNAME						X
MSGDB		X				
MSGNF		X				
OPTIONS	X	X	X	X	X	X
ORID				X		
PFKDATA						X
PFKLIT						X
PFKNUMB						X
PGMID				R	R	
PRINTER						X
SEGMENTS	X	X				
SPECIAL			X			
STX			X			
TRXID			R			
TSEGS			X			
USRLANG						D

Figure A-8 (Part 2 of 2). Summary: GENERATE Statement (Nonconversational)

R U L E S

	Segment Layout	Segment Handler	Input Transaction	Batch Driver Link Edit
	SEGL	SEGH	BAIT	BDLE
ADFID				D
AGROUP			X	
DBID	D	D	D	
KANAME			X	
LRULE			X	
PCBNO			D	
SYSID	R	R	R	R

Figure A-9. Summary: SYSTEM Statement (Batch)

R U L E S

	Segment Layout	Segment Handler	Input Transaction	Batch Driver Link Edit
	SEGL	SEGH	BAIT	BDLE
BYTES	R			
DBID	D	D	D	
DISPLAY			D	
ID	R	R		
KASCEND			D	
KEYNAME		R		
LENGTH	R			
MODE			D	
NAME		R		
NEWID	X	X		
OVERRIDE	X	X		
PARENT		R	R	
PCBNO			D	
SQLDIST		X		
SQLIND	X			
SQLISRT		X		
SQLNAME		X		
SQLUPD		X		
TRAILER	D		D	
TYPE	D		D	

Figure A-10. Summary: SEGMENT Statement (Batch)

R U L E S

	Segment Layout SEGL	Segment Handler SEGH	Input Transaction BAIT	Batch Driver Link Edit BDLE
AFA			X	
ASTATUS	X		X	
AUDIT	X			
BITOFF	D		D	
BYTES	R	R	R	
CAUDIT	X			
COFIELD			X	
COMPARE		X		
DECIMAL	D		D	
DISPLAY			X	
FAUDIT			X	
FLDOL			X	
FLDPOS			R	
ID	R			
ILENGTH			X	
ITYPE			X	
KEY		R	X	
KEYBLD			X	

Figure A-11 (Part 1 of 2). Summary: FIELD Statement (Batch)

R U L E S

	Segment Layout	Segment Handler	Input Transaction	Batch Driver Link Edit
	SEGL	SEGH	BAIT	BDLE
LENGTH	R	R	R	
MODE			X	
MSG	X			
NAME	R			
OFFSET	X			
PAUDIT			X	
POSITION	D			
REDEFINE	X			
REQUIRED			X	
SDECIMAL			D	
SIGN	D		D	
SQLISRT		X		
SQLNAME		X		
SQLNULL	X			
SQLORD		X		
SQLUPD		X		
TYPE	D		D	
VALUE		X		

Figure A-11 (Part 2 of 2). Summary: FIELD Statement (Batch)

R U L E S

	Segment Layout	Segment Handler	Input Transaction	Batch Driver Link Edit
	SEGL	SEGH	BAIT	BDLE
AEXIT				X
AGROUP			D	
ALANG				X
ALIAS	X	X		
BYPASS			X	
CHKPT				X
CNT			X	
DBPATH			X	
DEXIT				X
DLANG				X
DLET			X	
EOF			X	
FREQ				X
ISRT			X	
KANAME			X	

Figure A-12 (Part 1 of 2). Summary: GENERATE Statement (Batch)

R U L E S

	Segment Layout SEGL	Segment Handler SEGH	Input Transaction BAIT	Batch Driver Link Edit BDLE
LANGUAGE			D	D
LRULE			X	
MSGDB		X		
MSGNF		X		
OPTIONS	X	X	X	X
PGMID				R
PHEADING				D
SEGMENTS	X	X		
SHTABLE				X
SLRTABLE				X
SPECIAL			X	
SPNAMES				X
SPTABLE				X
TRXID			R	
TSEGS			X	
WTOMSG				D
WTORMSG				D

Figure A-12 (Part 2 of 2). Summary: GENERATE Statement (Batch)

APPENDIX B. DATA CONVERSIONS AND MAPPINGS

Conversion and mapping of data between different data types can occur at various points within IMSADF II. This appendix explains what to expect when these conversions and mappings are performed.

The points where the conversions or mappings take place in IMSADF II are:

- Data input and output during conversational processing
- Data input in nonconversational and batch processing
- Data output as directed by an Output Format Rule
- Data mapping to and from special processing routines

The following matrix and associated notes can be used to determine the result of a conversion/mapping operation. The nine rows represent input (FROM) fields and the nine columns represent output (TO) fields. The numbers in the matrix refer to the associated notes. These notes explain what happens on a particular combination of FROM and TO data types.

Conversational screen input and output conversions are sometimes handled differently from other conversion and mapping operations. These screen input and output fields are considered to be alphanumeric. Any differences that exist are described in specific notes following the matrix.

Invalid input data will result in error messages being generated for the fields in error. Only online processing gives the user the capability to correct the data immediately.

If invalid data is detected in mapping for special processing, a return code of 4 is given to the calling special processing program.

Data exceptions can occur if invalid packed decimal data is converted from a display format or if invalid packed decimal data is mapped to or from a special processing routine.

FROM Field Type	TO Field Type								
	A	N	AN	D	DT	PD	B	BT	X
A	1,2	3,10 11	1,2	3,4 10,11	1,2	3,4 11	3,4	20	N/A
N	1,2	3,10 11	1,2	3,4 10,11	1,2	3,4 10	3,4 10	20,21	N/A
AN	1,2	3,10 11,12 13,26	1,2	3,4 10,11 13,26	3,16	3,4 10	3,4 10,13 13,26	20	24
D	5,6	3,4 10,11	5,6	3,4 10,11	1,2	3,4 10	3,4 10	20,21	N/A
DT	2,7	3,10 11	2,7	3,4 10,11	1,2	3,4 10	3,4 10	20	N/A
PD	5,6	3,4 6,11	5,6	3,4 6,11	2,17	3,4	3,4 6	6,20 21	N/A
B	5	3,4 11	5	3,4 11	2,17	3,4	3	21	N/A
BT	8	18	8	18	N/A	18	18	23	N/A
X	2,9	N/A	2,9	N/A	N/A	N/A	N/A	N/A	1,25

DATA TYPES:
A—Alpha
N—Numeric
AN—Alphanumeric
D—Decimal
DT—Date
PD—Packed Decimal
B—Binary
BT—Bit
X—Hexadecimal

N/A—Not a valid FROM and TO combination

* Decimal and Packed Decimal fields may be specified as signed or unsigned (SIGN=Y or N) through the Rules Generator, with the following results:

- If a TO field is Decimal or Packed Decimal and SIGN=Y, the resultant sign will be 'C' for plus or 'D' for minus.
- If SIGN=N, the sign will always be 'F'.
- If a minus number is moved into an unsigned field, the minus sign is lost.
- Numeric fields are always unsigned.

Figure B-1. Conversion and Mapping Matrix

The numbers in the matrix refer to the following notes.

Notes:

1. FROM field is left justified in the TO field.
2. TO field is truncated or padded with blanks if its size is different than the FROM field.
3. FROM field is right justified in the TO field. Conversion takes place if required. High order digits are truncated if the TO field is shorter than the FROM field.
4. Decimal positions are adjusted to the TO field specifications.
5. Data is converted to an edited format. Leading zeros are suppressed and a floating minus sign is inserted for negative numbers. The TO

field is filled with asterisks (*) if it is too small to contain the result.

6. A data conversion exception may occur if invalid packed data is in the FROM field.

7. FROM field (YYMMDD) converted to the TO field (MM/DD/YY or DD/MM/YY if optional change made at installation time).

8. A '0' or '1' is placed in the first byte of the TO field.

9. Each byte in the FROM field is converted to two characters in the TO field ('0'-'9', 'A'-'F').

10. FROM field is verified for numeric data and placed in the TO field. Conversion takes place if required.

11. TO field is filled with zeros on the left if longer than the converted result of the FROM field.

12. Numeric attribute is set for conversational input.

13. FROM field is in edited format. It may have leading or trailing blanks, a leading minus sign, and one decimal point.

14. FROM field is verified for numeric data before conversion to the TO field.

15. Omitted.

16. FROM field (MM/DD/YY or DD/MM/YY) is converted to the TO field (YYMMDD).

17. The integer portion of the FROM field is converted to YYMMDD and is stored (left justified) in the TO field.

18. TO field is set to the value 0 or 1 depending upon the bit setting in the FROM field.

19. Omitted.

20. Bit in TO field is set to 0 or 1 depending upon the value in the FROM field (0 or 1). An error is indicated if the FROM field is not 0 or 1.

21. TO field bit is set to 1 if the value of the FROM field is 1. TO field bit is set to 0 if the value of the FROM field is 0. Decimal positions are ignored in the FROM field.

22. Omitted.

23. TO field bit is set to the value of the FROM field bit.

24. Characters are converted two at a time from the FROM field to create a hexadecimal character in the TO field. Conversion takes place from left to right. Invalid hexadecimal characters will be treated as zeros in nonconversational and batch processing and will produce an error message in conversational processing.

25. TO field is padded on the right with X'00' if required.

26. TO field is too small; an error condition is given if appropriate.

Examples

Several examples are given here to show exactly what will happen on conversions of various types. The decimal and packed decimal examples show the decimal point for clarification only; it is not actually in the field. Leading zeros are shown as they actually are in the numeric and decimal type fields. A blank is denoted by **b**.

The following field types and values are used in these examples along with other receiving fields that follow the same symbolic naming convention (type, length, and decimal places).

Symbol	Field Type	Length in Bytes	Decimal Places	Value
B4	binary	4	-	1234
D50	decimal	5	0	12345
D52	decimal	5	2	543.21
N5	numeric	5	-	12345
P50	packed[1]	5	0	12345
P52	packed	5	2	543.21
BIT	bit	-	-	1
DATE	date	6	-	770928

FROM Field	TO Field	Resulting Value	FROM Field	TO Field	Resulting Value
B4	D50	01234	D50	B4	12345
B4	D52	234.00	D50	D52	345.00
B4	N5	01234	D50	D71	012345.0
B4	P50	1234	D50	N5	12345
B4	P52	1234.00	D50	P50	12345
B4	DATE	001234	D50	P52	12345.00
			D50	DATE	12345b
D52	B4	543	N5	B4	12345
D52	D41	543.2	N5	D40	2345
D52	D50	00543	N5	D51	2345.0
D52	N5	00543	N5	P50	12345
D52	P50	543	N5	P52	12345.00
D52	P52	543.21	N5	DATE	12345b
D52	DATE	54321b			
P50	B4	12345	P52	B4	543
P50	D40	2345	P50	D40	0543
P50	D32	5.00	P52	D72	00543.21
P50	N7	0012345	P52	N4	0543
P50	P52	12345.00	P52	P21	43.2
P50	DATE	012345	P52	P32	543.21
			P52	DATE	000543
BIT	B4	1	DATE	B4	770928
BIT	D50	00001	DATE	D50	70928
BIT	D52	000.01	DATE	N7	0770928
BIT	P50	1	DATE	P50	770928
BIT	P52	.01	DATE	P52	770928.00

Figure B-2. Data Conversion Examples

---

[1] In PL/I, the declare must specify 8 or 9 digits to get 5 bytes reserved.

APPENDIX C. NAMING CONVENTIONS

```
*****************************************************************************
*                                  SUBSTITUTION CODES/COMMENTS(ADDITIONAL   *
* NAME FORMAT     ITEM NAMED                          INFO IN KEY AT BOTTOM) *
*****************************************************************************

ssssPM     *   POMR                  NAME OF A PRIMARY OPTION MENU RULE

ssssSMSS   *   SOMR(SMRULE)          NAME OF A SECONDARY OPTION MENU RULE

ssMOKxxn        "  (PRIMO)            NAME OF A PRIMARY OPTION SCREEN MOD
                                      n=SCREEN NUMBER

ssMOTKSa        "  (SECMO)            NAME OF A SECONDARY KEY SELECTION
                                      SCREEN MOD

ssssSxx         "  (SEGNM)            NAME OF A SEGMENT HANDLER RULE

*****************************************************************************
*  THE FOLLOWING INFORMATION DESCRIBES NAMES FOR CONVERSATIONAL INPUT       *
*  TRANSACTION RULES (ITR)                                                  *
*****************************************************************************

sssspgtt   *   ITR(CONV)             NAME OF A CONVERSATIONAL ITR
                                      p=PROJ CODE  g=GROUP CODE
                                      tt=TRANSACTION ID

ssMOStta        " (ITRMOD)            NAME OF SEG DISPLAY SCREEN AND ALSO
                                      SEG DISPLAY SCREEN MOD NAME

sspgttOa        "    "                SEG DISPLAY SCREEN MOD NAME WHEN
                                      DIFFERENT PG'S ARE ACCESSING THE SAME
                                      SCREEN

ssORxxO1        "    "                NAME OF OFR FOR SECONDARY TRANSACTION
                                      OR ALTERNATE OUTPUT MESSAGE
                                      xx=OFR ID

ssMOKtta        "    "                NAME OF PKS SCREEN AND ALSO NAME OF
                                      PKS SCREEN MOD

ssxxddnn        " (ITRPATH)   NAME OF A PATH OF ITR SEGS THAT COMPRISE A DBPATH
                              xx=SEG ID FOR TARGET OF THIS PATH
                              dd=DB ID FOR DB THAT CONTAINS THIS PATH
                              nn=SEG ID TRAILER

ssxxddnn        " (ITRSEG)    NAME OF A SEGMENT REQUIRED TO PROCESS A TRANSACTION
                              dd=DB ID FOR DB THAT CONTAINS THIS SEGMENT
                              nn=SEGMENT ID TRAILER

ssxxzzzz        " (ITRFLD)    NAME OF A FIELD WHICH IS IN AN ITRSEG
                              zzzz=FOUR-CHARACTER FIELD NAME

ssssyyyy        " AUDGP       NAME OF AUDIT GROUP TO ACCESS FOR THIS
                              TRANSACTION.  yyyy=AUDIT GROUP NAME AS
                              SPECIFIED IN THE 'AGROUP=' OPERAND IN A RULE
                              GENERATION SYSTEM STATEMENT.
ssxxzzzz        " FFRUL       NAME OF FIELD FOR WHICH FIELD FORCE AUDITS
                              ARE TO BE PERFORMED
ssMOVxxa       TEXT UTILITY SCREEN MOD NAME
                              xx=TEXT SEG ID
```

```
*****************************************************************************
*     THE FOLLOWING INFORMATION DESCRIBES NAMES FOR NONCONVERSATIONAL AND   *
*     BATCH ITR.                                                            *
*****************************************************************************


ssss#0tt  *   ITR(NONCONV)          # NAME OF NONCONVERSATIONAL ITR.

  (OR)


ssssBtt   *   ITR(BATCH)            B NAME OF BATCH ITR.

ssORttOa  *   '' (ITRMOD)     NAME OF THE OFR AND IF IT'S AN MFS MSG THEN
                              IT ALSO IS THE NAME OF THE OFR MOD.
                              THIS CAN BE OVERRIDDEN.  SEE NEXT ITEM.


yyyyyyyy      '' (ITRMOD)     NAME OF THE OFR AND OFR MOD IF THE ABOVE
                              DEFAULT IS OVERRIDDEN BY THE
                              'MODNAME=yyyyyyyy' OPERAND ON THE RULES
                              GENERATION GENERATE STATEMENT FOR A
                              NONCONVERSATIONAL TRANSACTION.


ssxxddnn      '' (ITRPATH)      --|
ssxxddnn      '' (ITRSEG)         |  SEE CORRESPONDING
ssxxzzzz      '' (ITRFLD)         |->ITEMS ABOVE FOR
ssssyyyy      '' AUDGP            |  CONVERSATIONAL ITR
ssxxzzzz      '' FFRUL          --|

*****************************************************************************
*     THE FOLLOWING INFORMATION DESCRIBES NAMES FOR SEGMENT LAYOUT          *
*     RULES (SLR).                                                          *
*****************************************************************************

ssssSRxx  *   SLR                   NAME OF A SEGMENT LAYOUT RULE
                                    'SR' MEANS IT'S A DB OR PS SLR
ssssURxx  *                         'UR' MEANS IT'S A DATA MAPPING SLR


ssxxzzzz       " FLDRL              NAME OF A FIELD

*****************************************************************************
*                              OTHER RULES                                  *
*****************************************************************************
ssssSyy   *   SHR                   NAME OF A SEGMENT HANDLER RULE
                                    yy=TARGET SEG ID


ssORxxOa *    OFR (STXRL)           NAME OF AN OFR AND IF THE OFR IS FOR
                                    AN MFS MESSAGE THEN IT ALSO IS THE
                                    NAME OF THE OFR MOD
                                    xx=OFR ID

ssssBRuu      BATCH DRIVER RULE     uu=TWO-CHARACTER BATCH DRIVER ID

sspgtt##  *   CCLM                  NAME OF A CONVERSATIONAL
                                    COMPOSITE LOAD MODULE
                                    tt=TWO-CHARACTER TRANSACTION CODE

ss#Itt##  *   NCLM                  NAME OF A NONCONVERSATIONAL COMPOSITE
                                    LOAD MODULE

ssssbbbb  *   SAR                   NAME OF A STATIC AUDIT RULE
```

```
*************************************************************************
*                                  KEY                                  *
*************************************************************************
a       REPRESENTS A ONE-CHARACTER MFS TRAILER.  IT IS OBTAINED FROM IMSADF II
        SYSTEM GENERATION JOB TWO UNLESS OVERRIDDEN BY THE 'STRAILER='
        OPERAND ON RULESGEN SYSTEM STATEMENT.  THE DEFAULT IS 1.
bbbb    REPRESENTS A FOUR-CHARACTER AUDIT GROUP CODE.  IT IS OBTAINED
        FROM THE 'AGROUP='  OPERAND ON A GENERATE OR SYSTEM STATEMENT
        IN RULES GENERATION.
tt      REPRESENTS A TWO-CHARACTER TRANSACTION CODE OBTAINED FROM THE
        'TRXID=' OPERAND ON A GENERATE STATEMENT IN RULES GENERATION.
pg      REPRESENTS A TWO-CHARACTER PROJECT/GROUP CODE OBTAINED FROM THE
        'PGROUP=' OPERAND ON A SYSTEM OR GENERATE STATEMENT IN RULES
        GENERATION.
seq id  REPRESENTS A TWO-CHARACTER SEQUENCE ID GENERATED BY THE
        RULES GENERATOR.
ssss    REPRESENTS THE FOUR-CHARACTER APPLICATION SYSTEM ID.
        IT IS OBTAINED FROM THE 'SYSID=' OPERAND IN RULES GENERATION.
ss      REPRESENTS THE FIRST TWO CHARACTERS OF THE FOUR-CHARACTER
        APPLICATION SYSTEM ID MENTIONED ABOVE.
xx      REPRESENTS A TWO-CHARACTER SEGMENT ID UNLESS OTHERWISE STATED.
        IT IS OBTAINED FROM THE 'ID=' OPERAND IN A RULES GENERATION
        SEGMENT STATEMENT UNLESS 'OPTIONS=OMFS' IS CODED, IN WHICH CASE
        IT IS OBTAINED FROM THE 'ORID=' OPERAND ON THE GENERATE STATEMENT.
zzzz    REPRESENTS A FIELD IDENTIFICATION OBTAINED FROM THE 'ID=' OPERAND
        IN A RULES GENERATION FIELD STATEMENT.
CLM     ABBREVIATION FOR COMPOSITE LOAD MODULE.
CONV    ABBREVIATION FOR CONVERSATIONAL.
ITR     ABBREVIATION FOR INPUT TRANSACTION RULE.
MOD     ABBREVIATION FOR MESSAGE OUTPUT DESCRIPTOR.
OFR     ABBREVIATION FOR OUTPUT FORMAT RULE.
PS      ABBREVIATION FOR PSEUDO SEGMENT.
SAR     ABBREVIATION FOR SEGMENT AUDIT RULE.
SEG     ABBREVIATION FOR SEGMENT.
SLR     ABBREVIATION FOR SEGMENT LAYOUT RULE.
TP      ABBREVIATION FOR TELEPROCESSING.
NOTES:
        UNDERLINED CHARACTERS IN NAMES ARE CONSTANTS.

        FOR THE VARIABLE PARTS OF NAMES, THE USER SHOULD BE CAREFUL NOT
        TO CHOOSE VALUES THAT WOULD CAUSE TWO DIFFERENT ITEMS TO HAVE
        THE SAME NAME.  THE USER SHOULD NOT USE THE CONSTANTS IN THE
        ABOVE NAMES AS VALUES FOR ANY OF THE VARIABLES.

        EXAMPLE 1.

        IF ONE CHOOSES 'SR' FOR A PROJECT GROUP THEN THE SLR (ssssSRxx)
        AND THE ITR (sssspgxx) WOULD BOTH BE 'ssssSRxx' FOR A
        CONVERSATIONAL TRANSACTION.

        EXAMPLE 2.

        IF 'SM' WAS USED FOR PROJECT GROUP AND 'SS' FOR SEGMENT ID THE
        SOMR (ssssSMSS) AND THE ITR (sssspgxx) WOULD BE 'ssssSMSS'.

        EXAMPLE 3.

        IF ONE CHOOSES 'pg' FOR PROJECT GROUP, 'tt' FOR TRANSACTION ID,
        AND 'pgtt' FOR AUDIT GROUP THEN BOTH THE ITR AND THE SAR (IF
        CREATED) WILL BE NAMED 'sssspgtt'.  THIS CAN CAUSE A PROGRAM
        TO LOAD THE ITR THINKING IT IS LOADING THE SAR.

        THOSE ITEMS MARKED WITH AN ASTERISK ARE LOAD MODULES FOUND IN
        RULES.LOAD.
```

APPENDIX D. ADDITIONAL APPLICATION SUPPORT TECHNIQUES

PROCESSING VARIABLE LENGTH SEGMENTS

A variable length segment (VLS) is a data base segment defined in a DBD (Data Base Definition) with a minimum and maximum length. The first two bytes of the segment specify the current length of the segment. It is a function of the application to maintain this length field during the replacement or insertion of the segment in the data base.

1. The maximum length of the VLS is specified in the Rules Generation Definition with BYTES=MAX or LENGTH=MAX.

2. The length field is specified:

   FIELD ID=fieldid,POS=1,TYPE=BIN,LEN=2

3. If the VLS is not the target of a path and DL/I path calls normally would be used to retrieve and update the segment, path calls should be disabled. This is accomplished by changing the corresponding data base PCB processing option. Instead of PROCOPT=AP on the PCB statement, PROCOPT=A should be specified. Note that all path call processing for that data base is not disabled. If the VLS is not used in all Input Transaction Rules against that data base, it is suggested that a duplicate PCB be set up (one with path calls on and one without). In this way, Input Transaction Rules not using the VLS can specify a PCBNO pointing to the path call PCB and make more efficient use of DL/I call logic. Respecification of the PCBNO operand can be achieved using the Rules Generator SEGMENT override technique.

Variable length segments can be handled by Special Processing routines and in some cases by Audit specifications. Following are two examples that can be processed by Audit logic:

CASE 1: The VLS has a repeating field that is 10 bytes in length and occurs a maximum of three times. Since IMSADF II does not support array processing, each recurring field must be defined as a uniquely named field. A PRELIM (or PRE) Audit will be set up to initialize the fields not retrieved, since the maximum length has been defined in the Rules Generator statements. A PROCESS (or POST) Audit will always be set up to calculate a new length. Rules Generator statements follow:

```
SEGMENT     ID=VS,LEN=38 ...

 FIELD      ID=VLEN,POS=1,TYPE=BIN,LEN=2

 FIELD      PAUD=YES, FAUD=YES, AUDIT=YES
            ID=KEY,POS=3,LEN=6,KEY=YES

 FIELD      ID=FLD1,POS=9,LEN=10

 FIELD      ID=FLD2,POS=19,LEN=10

 FIELD      ID=FLD3,POS=29,LEN=10
```

Audit Logic follows:

```
SYSID   =   TEST
AGROUP  =   YYYY
SEGID   =   VS
FIELD   =   VLEN

PRELIM
P1
*If seg VS is retrieved, clear variable fields if necessary
IF  RETRIEVAL = 'VS'
    IF VLEN LT 38
      TEVSFLD3 = ' '
```

```
        IF VLEN LT 28
          TEVSFLD2 = ' '
        ENDIF
      ENDIF
  ENDIF

  PROCESS
  P1
  IF TEVSFLD3 CHANGED = ON
      VLEN = 38
  ELSE
      IF TEVSFLD2 CHANGED = ON
          VLEN = 28
      ELSE
          VLEN = 18
      ENDIF
  ENDIF
```

CASE 2: The VLS has a one-byte record code that indicates the segment length and segment layout. Assume the segment has two layouts: 1) record code is 1 and length is 50 and 2) record code is 2 and length is 40. Two segment definitions are necessary since there will be different screen displays and a different layout of the fields in the segment. There will be key audits to switch to the proper screen for the entered key and to construct a browsing (Secondary Key Selection) screen with segment layouts consistent with the selected TRXID.

Rules Generator statements follow:

```
*INVENTORY LOCATION SEGMENT LAYOUT
 SEGMENT      ID=AA,LENGTH=50,NAME=SEGMENT1,...
  FIELD       ID=VLEN,POS=1,TYPE=BIN,LEN=2,AFA=YES
  FIELD       ID=KEY,POS=3,LEN=10,KEY=YES,KAUDIT=YES
  FIELD       ID=CODE,POS=12,LEN=1          RECORD CODE
  FIELD       ID=INAM,LEN=10                NAME
  FIELD       ID=STRT,LEN=10                STREET
  FIELD       ID=CITY,LEN=10                CITY
  FIELD       ID=STAT,LEN=8                 STATE

*INVENTORY STATUS SEGMENT LAYOUT
 SEGMENT      ID=BB,LENGTH=40,NAME=SEGMENT1,...
  FIELD       ID=VLEN,POS=1,TYPE=BIN,LEN=2,AFA=YES
  FIELD       ID=KEY,POS=3,LEN=10,KEY=YES,KAUDIT=YES
  FIELD       ID=CODE,POS=12,LEN=1          RECORD CODE
  FIELD       ID=INAM,LEN=10                NAME
  FIELD       ID=STOC,TYPE=PD,LEN=5         STOCK
  FIELD       ID=ONOR,TYPE=PD,LEN=5         ON ORDER
  FIELD       ID=DELV,TYPE=NUM,LEN=6        DELIVERY DATE
```

AA and BB define the same data base segment, but with different layouts.

```
GENERATE     TRXID=AA,OPTION=CVALL,DBPATH=AA,
             TRXNAME='INVENTORY LOCATION'
GENERATE     TRXID=BB,OPTION=CVALL,DBPATH=BB,
             TRXNAME='INVENTORY STATUS'
```

The above GENERATE statements define two different screens and Input Transaction Rules for the two usages of SEGMENT1. Audit logic for interrogating the keys and for setting the length field follow:

```
SYSID   =   TEST
AGROUP  =   YYYY
SEGID   =   AA
FIELD   =   KEY

KEY
P0
*CODE=2, SCHEDULE TRXID BE
    IF  TEAACODE = 2
        TRXID = 'BB'
    ENDIF
```

```
P1
*ONLY DISPLAY CODE=1    RECORDS ON SEC KEY SEL
IF TEAACODE = 2
    SKSDISP = OFF
ENDIF

FIELD   =   VLEN
PROCESS
P0
*If update trx set the length
    IF  MODE = '2,4,5'
        VLEN = 50
    ENDIF
```

The audit logic for the segment BB would use the same logic, except the CODE field would be checked for 1, the TRXID AA would be scheduled, and the BLEN would be set to 40.

PROCESSING A SECONDARY INDEX

A data base segment is processed through a secondary index by redefining the segment. Redefinition (or aliasing) of the segment consists of:

1. Assigning another 2 character ID to the indexed segment

2. Identifying the key fields according to the SRCH fields in the DBD (Data Base Description)

3. Specifying the KEYNAME operand equal to the XDFLD name in the DBD, and

4. Identifying the PCBNO operand as the PCB that specifies a processing sequence using the secondary index.

If any dependents to the secondary indexed segment are to be processed, those segments must also be aliased to redefine the parentage to the secondary indexed root segment. This allows the appropriate SSAs (Segment Search Arguments) to be built by the Rules Generator.

Note that the Secondary Key Selection requirement for unique keys continues to apply. The use of /SX in the DBD does not give application level uniqueness; only unique key values will give application level uniqueness. If non-unique keys are processed, the Secondary Key Selection screen will display the non-unique keys. However, if a selection is made, the first occurrence is always selected; or if the next screen of 18 occurrences is requested, the same screen will be displayed. Two examples of secondary indexing follow:

CASE 1: Secondary Index target and source segments are the same segment.

A part segment is sequenced by a part number key (PARTNBR) and the alternate search field is the part name (PARTNAME).

```
  DBD     NAME=DBDAM,ACCESS=...
  DATASET ...
  SEGM    NAME=PARTROOT,PARENT=0,BYTES=100
* PRIMARY KEY FIELD
    FIELD NAME=(PARTNBR,SEQ,U),BYTES=10,START=1
* SECONDARY KEY FIELD VIA SECIDX
    FIELD NAME=PARTNAME,BYTES=10,START=11
  .
  .
  .
* NAME PARTNAMX IS RULE GEN KEYNAME OPERAND
    XDFLD NAME=PARTNAMX,SEGMENT=PARTROOT,SRCH=(PARTNAME)
  DBDGEN

  RULE GENERATOR SOURCE.
  SYSTEM  SYSID=...
* DEFINE  PRIMARY SEG
  SEGMENT ID=PA,...,NAME=PARTROOT,KEYNAME=PARTNBR
    FIELD ID=KEY,KEY=YES,START=1,BYTES=10
* DEFINE ALIAS FOR SECIDX
  SEGMENT ID=PX,...,NAME=PARTROOT,KEYNAME=PARTNAMX,PCBNO=2
* KEYNAME MATCHES ZDFLD NAME, PCBNO NAMES
* PCB WITH ALTERNATE PROCESSING SEQUENCE
    FIELD ID=NAME,KEY=YES,START=11,BYTES=10
* KEY FIELD IS EQUAL TO SRCH FIELD ON XDFLD STATEMENT
* SEGMENT LAYOUT AND SEGMENT HANDLER RULES MUST BE
* GENERATED FOR EACH SEGMENT
  GENERATE SEGMENTS=(PA,PX),OPTION=SGALL
* GENERATE INPUT TRANSACTION RULE, PRIMARY KEY AND
* SEGMENT DISPLAY SCREENS
  GENERATE TRXID=PA,DBPATH=PA,
           TRXNAME='PRIMARY PART ACCESS',
           OPTION=CVALL
  GENERATE TRXID=PX,DBPATH=PX,
           TRXNAME='SECONDARY PART ACCESS',
           OPTION=CVALL
```

CASE 2: Secondary Index target and source segments are different segments

Assume that the Part Segment has a dependent segment that identifies inventory locations where this part is stored. There is an application requirement to identify a stored part, given an inventory location. To handle this situation, a secondary index is set up identifying the Part Segment as a secondary index target and the Inventory Location Segment as a source segment. Following is an example of how the DBD in the above example may be coded.

```
.
  DBD     NAME=DBDNAM,ACCESS=...
  DATASET ...
  SEGM    NAME=PARTROOT,PARENT=0,BYTES=100
* PRIMARY KEY FIELD
    FIELD NAME=(PARTNBR,SEQ,U),BYTES=10,START=1
  .
  .
  .
* TARGET IS PARTROOT AND SOURCE SEG IS INVENTORY
    XDFLD NAME=PARTLOCZ,SEGMENT=INVENTORY,SRCH=INVLOC
  .
  .
  .
  SEGM    NAME=INVENTORY,PARENT=PARTROOT,BYTES=50
    FIELD NAME=(INVLOC,SEQ,U),BYTES=8,START=1
  .
  .
  .
  DBDGEN
```

Rules Generator source to process the above.

```
  SYSTEM SYSID=,,,
  SEGMENT ID=PZ,BYTES=108,NAME=PARTROOT,KEYNAME=PARTLOCZ,PCBNO=3
* SEGMENT LENGTH (108)=100 (PARTROOT LENGTH) +9 (INVLOC SEARCH FIELD)
* KEYNAME IDENTIFIES XDFLD NAME
    FIELD ID=INVL,BYTES=8,START=101,KAUDIT=S,KEY=YES
* KAUDIT=S REQUESTS AUDIT INVOCATION AT SECONDARY
* KEY SELECTION TIME
  .
  .
  .
* SEGMENT RULES AND DISPLAY SCREENS
  GENERATE SEG=PZ,OPTION=SGALL
  GENERATE TRXID=PZ,DBPATH=PZ,
           TRXNAME='SECONDARY PART/INVENT LOC'
           OPTION=CVALL
```

The INVL field is the key field used in the SSA to access the PARTROOT segment, but is not part of that segment. Since INVL is not part of the segment, the Secondary Key Selection process, if invoked, will not have the INVL field value available. To fill this field, a key audit can be invoked at Secondary Key Selection time that would invoke an audit user exit. This audit user exit would move the PCB (Program Control Block) returned key feedback area to the audited field, INVL. This would then allow the Secondary Key Screen to display INVL values for selection. If any dependents are to be processed to the PZ segment, path calls must be turned off for this PCB. This is accomplished by specifying PROCOPT=A, rather than PROCOPT=AP.

An alternative approach to CASE 2 is to define the PARTROOT and INVENTORY segments according to their primary access and to define the secondary index as another data base segment with its own PCB. The secondary index segment can then be one data base path (DBPATH) and the PARTROOT and INVENTORY segments the other DBPATH. If enough data is carried in the secondary index, it can satisfy many Secondary Key Selection browsing requirements, as well as provide the input to build the keys for the PARTROOT and INVENTORY segments using the KEYBLD or KAUDIT functions.

PROCESSING SEGMENTS WITH NON-UNIQUE KEYS AND NO KEYS

IMSADF II provides the greatest function in standard data base processing mode when a unique key (in ascending sequence) exists for each data base segment. IMSADF II can also process (with less function) data base segments as part of the DBPATH with unique keys not in ascending sequence, non-unique keys, or search fields.

The Rules Generator requires a key field be defined for each data base segment. If this key is not a unique sequence key defined in the DBD, it is important for the developer to know the characteristics of the data and IMSADF II's processing strategy.

A discussion of processing strategy for cases other than segments with unique ascending sequence keys can be separated into four environments:

- **Segments Not in Ascending Sequence**

  Segments with unique keys can be processed by IMSADF II even if the keys are not in ascending sequence. An example is a root segment in a HDAM data base. If less than a complete key is provided for one of these segments, the Secondary Key Selection screen is presented to the operator. The segments are displayed on the screen in a physical sequential order. Generally, the keys will not be randomized in key sequence; therefore, the display is not in key sequence. A selection made from the Secondary Key Selection screen causes the proper root segment to be accessed.

  If a partial key is provided as input, it is unpredictable where the first segment will be physically retrieved from within the data base. IMSADF II will issue a GU > call for the partial key value. If no key value is provided (just >), an unqualified GU will be issued causing unpredictable results.

  If an attempt is made to access an additional segment on a subsequent browse screen, IMSADF II will issue a GU call for the last key value on the Secondary Key Selection screen followed by a GN call qualified on the root level provided KASCEND=NO was specified on the SEGMENT statement.

- **Segments with Non-Unique Keys**

  IMSADF II issues GU calls when retrieving segments in the DBPATH. Therefore, IMSADF II always retrieves the first physical occurrence of a segment when multiple segments at the same level have duplicate key values.

  The Secondary Key Selection (browse) function is not fully supported for segments with non-unique keys. When several segments with the same key value are displayed on the Secondary Key Selection screen, a selection of the second or subsequent duplicate key causes IMSADF II to retrieve the first segment occurrence with the same key value.

  Requesting a subsequent Secondary Key Selection screen may not provide proper results. If KASCEND=NO, IMSADF II will use the last key value to issue a GU call followed by a GN call. This situation may cause a loop situation with the same duplicate keys displayed on each Secondary Key Selection screen. If KASCEND=YES, a GU > call will be issued. This will cause segments to be skipped if the last key value had more duplicates.

- **Segments with Search Fields**

  One or more search fields may be defined as the key field if non-sequence keys exist in the segment. If multiple search fields are used as one IMSADF II key field, they must be contiguous in the segment. It is possible to have multiple definitions of the segment with each using different search fields as the key field.

  Secondary Key Selection takes on the same processing strategy as the previously described cases depending on the characteristics of the data in search fields (non-ascending or non-unique). If the search field makes each segment unique, normal IMSADF II primary key selection and automatic data base processing will work correctly.

- **Segments with No Fields in the DBD**

  IMSADF II cannot process segments in DBPATH if no fields are defined in the DBD. Limited processing capabilities are possible through Auditor DL/I calls. GN type calls can be performed to access the required segments. A dummy key field will have to be defined to allow the segment and field definitions to pass the Rules Generator checking.

APPENDIX E. ERROR MESSAGES - RULES GENERATOR

Following is a list of the error messages generated by the Rules Generator. Those messages which are not considered self-explanatory are followed by an expanded explanation of possible causes of the error. It is pointed out that, as with all compilers, the detection of an initial error can cause further processing of input to be done incorrectly and thus generate cascaded error messages. However, it is also possible in some instances to uncover additional errors which are not spurious and therefore, rather than terminating processing upon detection of the first error, the processing of the input statements is continued.

Diagnostic messages can occur during the definition and generation phases of Rules Generator execution. Once a diagnostic is encountered, the generation phase is no longer activated and a 'GENERATE NOT PERFORMED' diagnostic message will occur for each subsequent generation request. Definition diagnostics will continue to be created for the remainder of the input statements. Diagnostics 001 through 004 detect incorrect syntax. In those cases the question mark symbol, ?, is printed below the detected error along with the diagnostic message. Most of the other diagnostic messages will have a blank line between the incorrect statement and the diagnostic message. When a diagnostic refers to a previous segment or a field in a screen image, the segment ID or field ID and segment ID will be included as part of the diagnostic message.

The following is a list of messages in numerical order by message number:

ADFG001 PARAMETER OR VALUE SYNTAX ERROR

Explanation: The parameter or the operand value has invalid syntax. The error is normally caused by a missing parenthesis, a quote needed to close an operand value, or a missing comma needed to separate operands. For example:

```
DBPATH=(A2,A3,TRXID=XX
                   ?
ADFG001 PARAMETER OR VALUE SYNTAX ERROR
```

ADFG002 MISSING EQUAL SIGN

Explanation: An operand keyword is not followed by an equals operator. This usually means an operand value is not specified. For example:

```
LENGTH,KEY=YES
          ?
ADFG002 MISSING EQUAL SIGN
```

ADFG003 EXPECTED KEYWORD NOT FOUND

Explanation: A keyword is expected to follow the statement type keyword or a previous operand value. For example:

```
FIELD   ID=XX,16
              ?
ADFG003 EXPECTED KEYWORD NOT FOUND
```

ADFG004 EXPECTED CONTINUATION NOT RECEIVED

Explanation: An end of file was encountered where a continuation statement was expected. Check to see if a comma specified as the last character of the previous statement was valid or if a continuation statement was not supplied.

ADFG005 KEYWORD MISSING OR INVALID

Explanation: A keyword was not recognized as a statement name or was not valid for the statement type, or was misspelled. For example:

```
SYSTEM    ID=XXXX,PROJG=PG
```

should be

```
SYSTEM SYSID=XXXX,PGROUP=PG
```

May also occur if continuation is left out. For example:

```
SYSTEM  SYSID=XXXX,
SEGMENT ID=XX...
```

SEGMENT will be treated as a misspelled SYSTEM statement keyword. Correct the misspelling or supply a continuation statement as necessary.

ADFG006 VALUE NOT IN (keyword) TABLE (data)

Explanation: The operand value shown in the message (data) is not in the table of valid values for the specified keyword. For example:

```
          OPT=CONV
      ADFG006 VALUE NOT IN OPTION TABLE CONV
```

ADFG007 (keyword) NON-NUMERIC CHAR IN (data)

Explanation: The error message occurs when the value coded for a numeric keyword contains a non-numeric character (not 0-9). For example:

```
LENGTH=10.,
        ?
ADFG007 LENGTH NON-NUMERIC CHAR IN 10.
```

ADFG008 (keyword) INPUT DATA INCORRECT (data)

Explanation: The error message occurs when more values are coded than the keyword requires. For example:

```
SKLEFT='LINEONE',
SKLEFT='LINETWO',
SKLEFT='LINETHREE'
```

Operands that can result in this message are the following:

Keyword	Maximum Entries
ASRTABLE	40
DASRTAB	40
MAPTABLE	120
OFRTABLE	120
SHTABLE	120
SLRTABLE	120
SPTABLE	120
TPITTABL	120
BAITTABL	120
DELETE	40
TSEGS	100
DATACOMP	100
CURSOR	22
SEGMENTS	120
ISRT	100
DLET	100
DBPATH	100
SPNAMES	40
DEVNAME	40
DEVTYPE	40
Length or Bytes	255

ADFG009 (keyword) NULL VALUE OR BLANK

Explanation: The error message occurs when a value has been left out. For example:

```
SROW=(,6)
        ?
ADFG009 SROW NULL VALUE OR BLANK
```

ADFG010 (keyword) CHARACTER COUNT INVALID (data)

Explanation: The operand value shown in the message (data) contains too few or too many characters for the keyword. For example:

```
PGROUP=PG1
        ?
ADFG010 PGROUP CHARACTER COUNT INVALID PG1
```

Operands that can result in this message are the following:

Keyword	Required Count	Keyword	Required Count
ADFID	4	DELETE	2
AGROUP	4	TSEGS	2
PGROUP	2	DATACOMP	2
SOMTX	2	SEGMENTS	2
SYSID	4	ISRT	2
ASRTABLE	4	DLET	2
ITTABLE	2	DBPATH	2
OFRTABLE	2	DASRTAB	4
MAPTABLE	2	PGMID	2
SHTABLE	2	TRXID	2
SLRTABLE	2	SSI	4
SPTABLE	2	STRAILER	1
TPITTABL	2	DBID	2
BAITTABL	2	TRAILER	2
		ID(Segment)	2

ADFG011 (keyword) VALUE EXCEEDS ALLOWABLE (data)

Explanation: The operand value shown in the message (data) has exceeded the range of data allowed for this operand keyword. For Example:

```
FIELD   ID=AA,LENGTH=256
```

Operands that can result in the message are the following:

Keyword	Minimum Value	Maximum Value
BITOFF	0	7
COL	1	72
DECIMAL	1	13
DEVTYPE	2	5
FLDPOS	1	6000
LENGTH	1	255
LENGTH or BYTES (segment)	1	6000
LENGTH or BYTES	1	255
PCBNO	1	120
POSITION or START	0	6000
SCOL	1	132
SDECIMAL	1	13
SKSEGS	1	37
SLENGTH	1	255
SROW	6	43
CNT	1	25

ADFG012 (keyword) VALUE TRUNCATED (data)

Explanation: The data entered as the operand value for the keyword printed in the message has more characters than allowed. Operands that are associated with this diagnostic include the following:

KEYWORD	MAXIMUM
KEYNAME	8
KWNAME	8
NAME	8
PHEADING	60
SHEADING	54
SNAME	30
TEXT	30
TRXNAME	30
VALUE	30
WTOR	120
WTO	120

The user must shorten the name or text to the required maximum.

ADFG013 (keyword name) (keyword) MUST BE SPECIFIED

Explanation: The keyword printed in the message is a required keyword for the statement preceding this diagnostic. Required operands include the following:

Keyword Name	Requirement
system ID	SYSTEM statement
project group	GENERATE for Conversational transaction
SOMTX cluster	GENERATE for Conversational transaction
segment ID	SEGMENT definition
segment length	data base SEGMENT definition
segment name	data base SEGMENT definition
field ID	FIELD definition
field length	FIELD definition

ADFG015 MISSING SEGMENT OR MISSPELLED SEGID ID=(segid)

Explanation: Can be caused by a mistake in a screen image, a GENERATE statement or a FIELD statement with a segment of TYPE=MAP or TYPE=OUT.

1. In a screen image, a field has been defined as &nffff.xx where n is the mode. The segment xx is not defined by a SEGMENT statement in this input to the Rules Generator or, if it is defined, is not in a hierarchical path defined by the DBPATH operand or is not named in the TSEGS operand of the GENERATE statement for this transaction.

   Note: This message can be caused by the use of tabular definition of a screen image. In this case the segid in the error message may be garbled. The problem arises when some parts of a line are defined in tabular form with other parts in the &nffff.xx form. The solution is to define the whole line in tabular form in such cases.

2. On a GENERATE statement a segment ID is named but no SEGMENT statement with that ID appears in this input to the Rules Generator.

3. On a FIELD statement within a mapping segment or output segment (Output Format Rule) a SEGID operand names a segment ID but no SEGMENT statement of that ID appears in this input to the Rules Generator.

ADFG016 GENERATE NOT PERFORMED - ERROR COND

Explanation: As a result of some other error, no more rules or screens are being produced. Only syntax checking is being performed. This message should always be preceded somewhere in the source statement listing by another error message explaining the problem.

ADFG017 INVALID ASTATUS REQUEST CHARACTER. (data)

Explanation: None

ADFG018 SCREEN DATA CONTAINS INVALID ROWCOL ID=(trxid)

Explanation: None

ADFG019 KEY FIELD REQUIRED FOR DB SEGMENT ID=(segid)

Explanation: Every data base segment defined to IMSADF II must have a key field defined in it. It will be used to construct DL/I Segment Search Arguments and format the Key Selection screens.

Code the KEY=YES on a FIELD statement and KEYNAME=nnnnnnnn on the SEGMENT statement, where nnnnnnnn is the name to be quoted in the DL/I Segment Search Argument. The fields marked KEY=YES must be contiguous and the sum of their lengths must equal the length of the nnnnnnnn field. The user will be prompted (via Key Selection in Conversational processing) to enter values for each of these fields which will then be concatenated and used to retrieve the segment.

ADFG020 SYSTEM STATEMENT MUST BE SUPPLIED

Explanation: None

ADFG021 SEGMENT STATEMENT REQUIRED BEFORE FIELD

Explanation: None

ADFG023 SEGMENT DEFS/TRANSACTIONS EXCEEDS 200

Explanation: The Rules Generator can only handle a combined maximum of 200 SEGMENT statements and transaction requests in a single run. Larger systems can be generated by breaking them down into several input decks, each beginning with a similar SYSTEM statement having the same SYSID.

ADFG024 NUMBER OF FIELD STATEMENTS EXCEEDS PROGRAM CAPACITY

Explanation: By default, the Rules Generator can only handle 500 FIELD statements in a single run. Either split the deck as for message ADFG023 or increase the limit by coding PARM=(n,2) on the JCL EXEC statement where n is the number of FIELD statements to be allowed and 2 is another default concerned with screen image capacity (see message ADFG037).

ADFG025 &ENDR or &ENDS INPUT DELIMITER MISSING

Explanation: The &ENDS statement must begin in column 1 and must not be more than the number of lines on the smallest device specified from the start of the screen image. If physical paging is used (&=P) the &ENDS must still be coded at the end of the last page. If fields are defined in the tabular format, the tabular definitions must precede the &ENDS statement.

ADFG026 SECONDARY KEY SCREEN DATA FOR SEGMENT DOES NOT FIT ON ONE LINE.

Explanation: None

ADFG027 DEVICE TYPE NOT SPECIFIED FOR DEVNAME. (data)

Explanation: A screen generation request specified fewer DEVTYPE entries than DEVNAME entries.

ADFG028 TRXID REQUIRED FOR TRANSACTION REQUEST

Explanation: A GENERATE statement with OPTION=INTR,SEGD,KEYD must have a TRXID operand. For Conversational and Nonconversational composite rules load module link-edits (OPTION=CTLE,NTLE), this operand is required.

ADFG029 DBPATH/TSEGS REQUIRED FOR TRANSACTION REQUEST

Explanation: A transaction with a key selection path specified or assumed by default must specify one or more target segments. This means that DBPATH must be coded on any GENERATE statement with a Standard processing transaction request unless KSEL=NO is also coded.

ADFG030 INVALID SCREEN IMAGE MODE FOR FIELD. ID=(fieldid.segid)

Explanation: In a screen image definition a field is specified in the format &mffff.xx where m is the mode:

- 4-modifiable, modifiable following transaction mode 6
- 5-modifiable, displayed
- 6-non-modifiable, displayed
- 7-modifiable, non-displayed

In the tabular form of screen image definition, the mode (4,5,6 or 7) must be coded in one of the columns 31 to 36.

ADFG031 FIELD NOT FOUND IN SPECIFIED SEGMENTS. ID=(fieldid.segid)

Explanation: Fields defined in a screen image must also be defined by FIELD statements in the same Rules Generator run. If the field is defined in the screen image in the form ffff.xx (normal or tabular style) the FIELD ID=ffff must be in the SEGMENT ID=xx and this segment must be either named in the TSEGS or DBPATH operands or be the parent or higher level of a segment in the DBPATH operand of the GENERATE statement for the transaction associated with this screen image.

If the field is defined as ffff without the qualifying segment ID the field ID must be unique in this Rules Generator run. The segment containing the field must be specified or implied in the TSEGS or DBPATH operands as above.

This error may occur if COFIELD= xxxx.xx is used to refer to the same pseudo segment field from more than one Input Transaction Rule in a Rules Generator Run. COFIELD restriction are defined in the IMS Application Development Facility II Version 2 Release 2 Application Development Guide.

ADFG032 FIELD DISPLAY LENGTH OVERLAPS NEXT FIELD. ID=(fieldid.segid or adfname)

Explanation: The adfname may be SYSMSG or one of the other IMSADF II fields on the screen. If not specified in the screen image, SYSMSG will default to line 24, ACTION for paging (when required) to line 22 and the other IMSADF II fields to line 23. Any other fields that overlap with these default fields will cause this error message.

A common mistake is to put numbers in the screen image while using an editor. These appear on the screen as literals and may overlap with data fields.

ADFG033 **FIELD DISPLAY LENGTH NOT SUFFICIENT FOR DECIMAL, SIGN, AND VALUE. ID=(fieldid)**

Explanation: None

ADFG034 **KEY FIELD TYPE CANNOT BE (BIT/FLOAT). ID=(fieldid)**

Explanation: None

ADFG035 **LENGTH OF KEY FIELDS FOR SEGMENT EXCEEDS 70 BYTES. ID=(segid)**

Explanation: The length of the Key of a Segment displayed on Secondary Key Selection Screen must not be longer than 70 Bytes. See also the SKSEGS operand in the SEGMENT Statement.

ADFG036 **FIELD ALREADY SPECIFIED IN SCREEN IMAGE. ID=(fieldid.segid or adfname)**

Explanation: Note that fields begin with an attribute byte which is placed at the position of the ampersand or quote in front of a literal. The SYSMSG field is 70 bytes long. A field must fit on to one line; otherwise message ADFG032 is raised. Refrain from using the first position of the first line of the screen.

ADFG037 **IMAGE EXCEEDS AVAILABLE STORAGE**

Explanation: When using multiple physical pages for transactions, increase the number of image blocks to the number of pages required by the largest transaction by coding PARM=(500,N) on the JCL EXEC statement for the Rules Generator. By default the value of N is 2, meaning 2 main storage blocks of 3600 bytes. 500 is the default number of FIELD statements (see message ADFG024).

ADFG038 **PROGRAM ID REQUIRED FOR LINK EDIT**

Explanation: A GENERATE statement with OPTION=STLE, SPLE, SPEC, TPLE, NCLE or BAIT must have a PGMID operand. For standard processing link-edits (OPTION=STLE or NCLE) the value of PGMID is the cluster code. Thus for each different SOMTX operand value (i.e., cluster code, on the SYSTEM or GENERATE statements), such a link-edit is necessary. For the meaning of PGMID in special processing or batch link-edits refer to the manual.

ADFG039 **REFERENCE NAME NOT FOUND IN IMAGE. VCOL=(data)**

Explanation: None

ADFG040 **STX MESSAGE TYPE NOT SPECIFIED AS MFS or TRX**

Explanation: None

ADFG041 **STX VALUE IS INVALID - (data)**

Explanation: None

ADFG042 **ONLY 60 STX MESSAGES MAY BE ENTERED**

Explanation: None

ADFG043 **VALUE FOR KEYWORD IS INVALID. (KEYWORD=data)**

Explanation: None

ADFG044 **NO FIELD STATEMENTS FOR SEGMENT. ID=**

Explanation: The segment indicated by ID= has been defined without any field statements.

ADFG045 PARENT ID/NAME NOT FOUND. (data)

Explanation: None

ADFG046 LITERAL CANNOT START IN FIRST COLUMN OF PAGE.

Explanation: None

ADFG047 (keyword) FIELD REQUIRED IN SCREEN IMAGE. PAGE=

Explanation: The Segment Display Screen defined by SIMAGE requires this field on the designated SIMAGE page. This error is output only if Line 22, 23, and 24 of SEGMENT Display Screen is not available for this field to be displayed.

ADFG048 MORE THAN 22 SEGMENT DISPLAY PAGES REQUESTED

Explanation: None

ADFG049 IMAGE LINES EXCEED MAXIMUM LINES FOR DEVICE(S)

Explanation: None

ADFG050 ASSEMBLY DIAGNOSTIC ENCOUNTERED

Explanation: As a result of processing the source statements, the Rules Generator creates Assembly Language statements and dynamically invokes the Assembler. It may occasionally happen that the Assembler detects an error in the internally generated code. To see the Assembly listing, code ASMLIST=NOGEN on the Rules Generator SYSTEM statement. It is usually possible to bypass the problem by inspecting the Assembly statements and amending the Rules Generator source statements appropriately.

ADFG051 LINK EDIT DIAGNOSTIC ENCOUNTERED

Explanation: Most GENERATE statements cause the Rules Generator to invoke the Linkage Editor dynamically. Diagnostics may be encountered, particularly as a result of link-editing special processing programs or audit exit routines. The normal Linkage Editor listings and diagnostics are printed. The options used by the Linkage Editor can be set by coding (e.g., LOPTPARM='NCAL, XREF,LIST') on the SYSTEM statement.

ADFG052 RULE UPDATED BY DEFAULT

Explanation: A Secondary Option Menu Rule is a list of valid TRXIDs in the application system. By default, the GENERATE OPT=SOMSS statement will update the existing Secondary Option Menu Rule by merging in the TRXIDs generated in the current Rules Generator run.

ADFG053 RULE FOR UPDATE NOT FOUND IN STEPLIB

Explanation: Refer to the description of message ADFG052. Message ADFG053 is caused by coding UPDATE=YES on the GENERATE OPT=SOMSS statement when no rule has previously been generated. Initially, the rule should be created by coding UPDATE=NO. By default, the rule will be updated if it is present; otherwise, it will be created.

ADFG054 PROJECT GROUP REQUIRED FOR LINK EDIT

Explanation: A GENERATE statement with OPTION=CTLE,NTLE for composite rules load module link-edit requires a PGROUP operand to be entered.

ADFG055 NUMBER OF TRXS FOR SOM RULE EXCEEDS 300

Explanation: A GENERATE statement with OPT=SOMSS for an SOM rule has been requested with more than 300 IMSADF II transactions for a single system ID.

ADFG056 TRANSACTION SPECIFIED NOT FOUND IN SOM RULE. ID=

Explanation: None.

ADFG057 COFIELD NOT IN SEGMENTS DEFINED FOR TRANSACTION

Explanation: None

ADFG058 (KEYWORD) NOT ALLOWED TO BE OVERRIDDEN

Explanation: A field statement with a value for START,POSITION or BYTES which would result in changing the value of the previously defined field. This message can result if a user tries to define a new field within a segment with new values for START, POSITION or BYTES and selects a previously defined field ID. A merge will occur if the field ID matches a previously defined field ID and field parameters other than START, POSITION, or BYTES are used.

ADFG059 DBPATH REQUIRES THAT KEYSL=YES BE SPECIFIED

Explanation: The KEYSL operand is defaulted depending on DBPATH. If DBPATH is specified, the default value is KEYSL=YES; otherwise, the value is KEYSL=NO. IMSADF II transaction drivers require the KEYSL=YES if DBPATH is specified. This error is to prevent DBPATH and KEYSL=NO combination of keywords.

ADFG060 FIELD NOT ALLOWED ON SIGN ON. ID=

Explanation: The identified field is invalid for use in the screen image for the sign-on screen. See discussion of the SOIMAGE= operand for valid fields.

ADFG061 SIGNON ALREADY GENERATED

Explanation: A SIGNON screen has already been generated on the SYSTEM statement or a previous GENERATE statement. Since SOIMAGE was requested (in either place), different descriptions of the SIGNON screen may have been requested.

ADFG062 FIRST 4 POSITIONS OF LAST LINE NOT BLANK

Explanation: Positions 1 through 4 of the last screen line of screen image is reserved for the size field and not available to the user.

ADFG063 FIELD SPECIFIED FOR CURSOR POSITION NOT FOUND IN SIMAGE. ID= (FIELDID.SEGID)

Explanation: A field specified on the CURSOR subparameter in the SIMAGE parameter cannot be located on the screen image layout. For multi-page screen image, check the ID parameter against the CURSOR subparameter. If ID is the second parameter for CURSOR, then the error is on page two the screen image.

ADFG064 FIELD ID FOR REDEFINE NOT PREVIOUSLY DEFINED IN THIS SEGMENT. ID= (fieldid)

Explanation: None

ADFG065 OFFSET SPECIFIED ON REDEFINE EXCEEDS BASE FIELD LENGTH. ID= (fieldid)

Explanation: None

ADFG066 **UNABLE TO OPEN ADF EXTRACT DDNAME SPECIFIED. ID= (DDNAME)**

Explanation: None

ADFG067 **ERROR IN THE DB2 PRECOMPILER FOR TABLE - (tableid)**

Explanation: The DB2 precompiler found an error of severity greater than four while attempting to precompile the Table Handler Rule created for the table whose table-id is (tableid).

ADFG068 **MODULE FOR ADF EXTRACT PROCESSING CANNOT BE LOADED. ID= (ADF EXTRACT module)**

Explanation: None

ADFG069 **MODULE FOR DB2 PROCESSING CANNOT BE LOADED. ID= (DB2 module)**

Explanation: None

ADFG070 **SQLNULL NOT ALLOWED ON KEY FIELD. ID= (fieldid)**

Explanation: None

ADFG071 **LIBRARY I/O ERROR IN MEMBER**

Explanation: None

ADFG073 **CANNOT OPEN LIBRARY ADFLIB/IMAGELIB**

Explanation: None

ADFG074 **MEMBER NOT IN LIBRARY**

Explanation: None

ADFG075 **LIBRARY DIRECTORY I/O ERROR**

Explanation: None

ADFG076 **LENGTH OF FLOAT FIELD MUST BE (data) BYTES**

Explanation: All fields with TYPE=FLOAT must be 8 bytes in length.

ADFG077 **NUMBER OF COLUMNS ON PARM STATEMENT OF RULES GENERATOR JCL HAS BEEN EXCEEDED**

Explanation: The number of column statements has exceeded value specified for number of columns in the PARM parameter passed to the Rules Generator. The default value is 100 column statements.

ADFG078 **INVALID LENGTH IS SPECIFIED IN DBCS/MIXED FIELD**

Explanation: The length of a DBCS field should be 2-256 bytes and should be an even number of characters. For a mixed field, the length should be a minimum of 4 bytes.

ADFG079 **INVALID DBCS CHARACTER WAS FOUND IN SCREEN IMAGE**

Explanation: The valid range of DBCS characters is hex 4141 to hex FEFE. The hex value of 4040 is also valid.

ADFG080 **TWIN PROCESSING NOT VALID FOR A TABLE STATEMENT. ID= (segid)**

Explanation: None

ADFG081 **SQLNAME MUST BE ENTERED ON TABLE STATEMENT. ID= (segid)**

Explanation: None

ADFG082 **I/O ERROR IN WRITING TO ADF EXTRACT FILE. ID= (file id)**

Explanation: None

ADFG083 **SEGMENT SPECIFIED IS NOT A DB SEGMENT. ID= (segid)**

Explanation: Only database segments will be extracted for the Data Dictionary.

ADFG084 **THIS SEGMENT WAS SUCCESSFULLY PROCESSED BY ADF EXTRACT FOR THE DATA DICTIONARY: ID= (segid)**

Explanation: None

ADFG085 **TABLE (tableid) HAS NO DB2 COLUMN WITH SQLISRT=YES**

Explanation: There must be at least one DB2 column specified as SQLISRT=YES if the DB2 function INSERT is specified. The error was found while creating the Table Handler Rule for table (tableid).

ADFG086 **TABLE (TABLEID) HAS NO DB2 COLUMN WITH SQLUPD=YES**

Explanation: There must be at least one DB2 field specified as SQLUPD=YES if the DB2 function FUPDATE or CUPDATE is specified. The error was found while creating the Table Handler Rule for table (tableid).

ADFG087 **(db2function) IS NOT A VALID ADF/SQL FUNCTION**

Explanation: (db2function) was specified in the SQLCALL clause of Table Handler Rule GENERATE statement. The only valid functions are: CSELECT, INSERT, CUPDATE, CDELETE, KSELECT1, SELECT, UPDATE, DELETE, KSELECT2, DSQLCALL, and NONE.

ADFG088 **MODULE FOR NATIONAL LANGUAGE SUPPORT CANNOT BE LOADED. ID= (module)**

Explanation: Language suffix specified is not installed.

ADFG089 **UNABLE TO OPEN SQLH DDNAME SPECIFIED. ID=(DDNAME)**

Explanation: (ddname) is probably missing or incorrectly specified in the JCL for this Rules Generator run.

ADFG090 **(hvar) IS AN INVALID USER HOST VARIABLE**

Explanation: (hvar) was specified as part of a WHERE clause in user defined function. (hvar) is not in the proper format for a host variable. It should be of the form ':ffff.tt' where 'ffff' is the fieldid and 'tt' is the tableid.

ADFG091 **TABLE (tableid) OF HOST VARIABLE :(fieldid.tableid) IS MISSING**

Explanation: The (tableid) of this user host variable is not defined in the Rules Generator run.

ADFG092 **FIELD (fieldid) OF HOST VARIABLE: (fieldid.tableid) IS NOT IN TABLE (tableid)**

Explanation: The (field) of this user host variable is not defined as a DB2 field to table (tableid).

ADFG093 **(db2function) IS AN INVALID DB2/SQL USER FUNCTION**

Explanation: The user DB2 function (db2function) specified in columns 10 to 17 is not a valid user function. The only valid user functions are: SELECT, UPDATE, DELETE, CURSOR, OPEN, CLOSE, and FETCH.

ADFG094 TOO MANY USER DB2/SQL STATEMENTS

Explanation: The maximum number of user statements of twenty-six has been exceeded.

ADFG095 CURSOR NAME IS MISSING ON STATEMENT LABELED- (label)

Explanation: Cursor name not found in columns 19 to 71 of user statement whose label is (label). The user function is OPEN, FETCH, or CLOSE.

ADFG096 CURSOR NAME TOO LONG ON STATEMENT LABELED - (label)

Explanation: The cursor name is too long, more than eight characters. It was found in columns 19 to 71 of a user statement whose label is (label). The user function is OPEN, FETCH, or CLOSE.

ADFG097 THIS KEYWORD MAY NOT BE OVERRIDDEN. ID=(keyword)

Explanation: None

ADFG098 SQLNULL REQUIRES SQLIND AND TYPE=TBL ON TABLE STATEMENT

Explanation: None

ADFG099 DBRMLIB DD CARD IS MISSING OR INVALID

Explanation: Check the JCL for the DBRMLID DDcard.

ADFG100 NO DB2 COLUMNS FOR TABLE (tableid)

Explanation: There must be at least one DB2 column in each table specified in a GENERATE statement with the TABH option. (tableid) was specified but has no columns which contains SQLNAME parameter.

ADFG101 KSELECT1 OR KSELECT2 MUST HAVE A KEY, TABLE ID IS (tableid)

Explanation: KSELECT1 OR KSELECT2 has been specified for table (tableid), but (tableid) does not have a DB2 key field.

ADFG102 SQLNAME AND REDEFINE ARE MUTUALLY EXCLUSIVE KEYWORDS FOR DB2. ID= (fieldid)

Explanation: None

ADFG103 SQLUPD=YES CAN NOT BE SPECIFIED FOR A KEY=YES COLUMN. ID= (fieldid)

Explanation: None

ADFG104 (tableid) IS NOT A TABLE

Explanation: (tableid) did not specify TYPE=TBL on the SEGMENT/TABLE statement.

ADFG105 CAN HAVE ONLY ONE TABLE, IF SQLUSER=YES

Explanation: User functions only apply to one table. Thus a table handler GENERATE statement must not have both user functions and multiple tables.

Where '(keyword)' is specified, the operand in error will be the output. Where '(data)' is specified, the operand value in error will be the output. Where '(fieldid)', '(fieldid.segid)', '(segid)', or '(adfname)' is specified, the field or segment definition in error will be identified.

ADFG106 COLUMN (fieldid) OF TABLE (tableid) OVERLAYS A PREVIOUS COLUMN AT POSITION (position).

Explanation: DB2 does not support redefinitions of columns. Therefore the IMSADF II Rules Generator does not allow column definitions that overlay previous column definitions. You specify additional COLUMN/FIELD statements that are redefinitions of DB2 columns by not including the SQLNAME operand. Only COLUMN/FIELD statements that have a SQLNAME operand are treated as DB2 column definitions.

ADFG107 TABLE ID (tableid) HAS INVALID TABLE NAME FOR (function) FUNCTION.

Explanation: The SQLNAME operand on a TABLE statement can contain DB2 table names in four forms: unqualified, qualified, multiple names, and correlated name. An unqualified or a qualified table name is valid for SQL statements that can be included in a Table Handler Rule. Multiple names are used to define a view that represents a join of two or more DB2 tables. When specified, the normal DB2 restrictions for views are enforced. The view is not eligible for UPDATE, DELETE or INSERT. DB2 also restricts the use of correlated names when specifying the SQL insert function.

ADFG108 TEXT BEFORE IMBEDDED BLANK IN TABLE NAME FOR TABLE ID: (tableid) USED AS TABLE NAME FOR INSERT.

Explanation: The SQLNAME operand on the TABLE statement contains a DB2 correlated name in the form 'TABLE.NAME X'. DB2 does not allow a correlated name to be specified on a SQL INSERT function. The table name used for the SQL INSERT function in the Table Handler Rule disregards the correlated reference portion of the name. The table name used is of the form 'TABLE.NAME'. The entire correlated name specified in the SQLNAME operand is used for all other SQL statements included in the Table Handler Rule.

ADFG109 THE DSNAME FOR DBRMLIB MUST INCLUDE A MEMBERNAME

Explanation: The DB2 precompiler stores a DBRM member for each Table Handler Rule created into this catalogued MVS partitioned data set. If the JCL does not specify a member name the directory of the PDS is probably destroyed. For additional information reference the IMS Application Development Facility II Version 2 Release 2 DATABASE 2 Application Specification Guide, section "Using the Rules Generator Execution Procedure".

ADFG110 THE DISP PARM FOR DBRMLIB MUST BE OLD OR SHR

Explanation: Although the DB2 precompiler requires a member name on the JCL, the IMSADF II Rules Generator replaces the member name used on the JCL with a name that contains the IMSADF II sysid and table id. With this benefit comes the restriction that the DISP parameter on the JCL must be OLD or SHR. For additional information see the IMS Application Development Facility II Version 2 Release 2 DATABASE 2 Application Specification Guide, section "Using the Rules Generator Execution Procedure".

ADFG111 TOO MANY HOST VARIABLES IN USER SQL STATEMENT (label)

Explanation: IMSADF II allows a maximum of 100 distinct host variables in the WHERE clause of a user SQL statement.

ADFG112 MAPSET MEMBER CAN NOT BE OPENED IN CICSMAPS DATASET - (mapset name)

Explanation: Insure that a CICSMAPS DD statement defining a partitioned data set is in the JCL for the Rules Generator.

ADFG113 NUMBER OF SCREENS ON PARM PARAMETER IS GREATER THAN 22

Explanation: The second value of the PARM parameter in the JCL that invokes the Rules Generator is greater than 22. The Rules Generator only supports 1-22 pages of the Segment Display Screen.

ADFG114 NUMBER OF INPUT TRANSACTION FIELDS EXCEEDS MAXIMUM

Explanation: The number of fields in a multiple-page screen image has exceeded the maximum allowed. This applies only to the CICS/OS/VS environment. See note 2 on page 2-101 for more information.

APPENDIX F. ERROR MESSAGES - AUDIT LANGUAGE

Following is a list of the error messages generated by the Audit Language Processor. Those messages which are not considered self-explanatory are followed by an expanded explanation of possible causes of the error. It is pointed out that, as with all compilers, the detection of an initial error can cause further processing of input to be done incorrectly and thus generate cascaded error messages. However, it is also possible in some instances to uncover additional errors which are not spurious and therefore, rather than terminating processing upon detection of the first error, the processing of the input statements is continued.

ADFH801 WHILE EXPECTED FOLLOWING DO

Explanation: Every DO keyword must be followed by the WHILE keyword.

ADFH802 = SIGN EXPECTED FOLLOWING KEYWORD

Explanation:

ADFH803 LITERAL OR NUMERIC CONSTANT EXCEEDS 70 CHARACTERS

Explanation: The longest string of characters which may be defined in a spanned data descriptor is 70 characters excluding the parentheses which bound it. If possible, the language processor will generate spanned descriptors within this limitation. However a single literal which encloses more than 70 positions or a string of more than 70 numbers with no intervening blanks cannot be subdivided.

ADFH804 CONDITIONAL EXPRESSION INCORRECT

Explanation: In the process of analyzing a conditional expression, a sequence of expression elements was encountered which could not be recognized. The expression elements consist of audited and related field names, equal and relational operators, keywords, etc.. This error could occur if the condition making up the conditional expression were used or ordered incorrectly in the statement.

ADFH805 MOVE SYNTAX INCORRECT

Explanation: An assignment statement could not be recognized. As with error 804, this error can occur if current statement was thrown out of phase because of a previous error. Other conditions causing this error are statements in which the elements were either used incorrectly or did not appear in a recognizable sequence.

ADFH806 LABEL NOT FOUND IN BRANCH TABLE

Explanation: Either an explicit GOTO label cannot be found or the erroneous use or deletion of an ENDDO or ENDIF prohibits the Rules Generator from assigning a proper line number to the NEXT TRUE or NEXT FALSE fields. The error may be caused by a previous statement.

As with message ADFH837, this error can occur if any leg contains more than 400 branch labels. In this case, the 401st branch target label would be a compiler generated IF statement (i.e., audit operation descriptor '16' or 'F2').

ADFH807 ENDIF DOES NOT CLOSE IF STATEMENT

Explanation: An ENDIF statement is not preceded by a corresponding IF statement.

ADFH808 ENDDO DOES NOT CLOSE DO STATEMENT

Explanation: An ENDDO statement is not preceded by a corresponding DO WHILE statement.

ADFH809 MOVE OF LIST OR RANGE INVALID

Explanation: An assignment statement which attempts to move a list of items or a pair of range values is invalid.

ADFH810 CLOSING PARENTHESIS AT END OF RELATED FIELD NAME IN AEXIT STATEMENT MISSING

Explanation: In the AEXIT statement, the use of parentheses to bracket the related field name is optional. If they are used, this error indicates that the closing parenthesis could not be found.

ADFH811 P2 LEG IS INVALID IN KEYAUDIT PHASE

Explanation: Only the P0 and P1 legs can be specified.

ADFH812 MAXIMUM LABEL VALUE EXCEEDED

Explanation: The number of generated labels exceeds either 99 or 1225 depending upon whether the XPANDLBLS option was specified.

ADFH813 ERRORMSG NOT VALID IN P2 LEG

Explanation: The ERRORMSG statement is limited to the P0 and P1 legs of the Audit Data Base.

ADFH814 INFOMSG ONLY VALID IN P2 LEG

Explanation: The INFOMSG statement cannot be used in the P0 or P1 legs of the Audit Data Base.

ADFH815 MESSAGE OR LABEL NUMBER NOT A NUMERIC VALUE

Explanation: This message is used for both message numbers which must be one to four digits or the STRTLBL statement has been used to specify a two character numeric value with which the generated labels are to begin but the label is not valid. The XPANDLBL option must be off.

ADFH816 THIS STATEMENT INVALID IN SUBROUTINE OR IF SYSID = COMMON

Explanation: In order for certain operation descriptors to function when an audited field name is specified, it is necessary convert its name into a related field format. This is done by concatenating the first two SYSID characters with the SEGID and the audited field name and treating the resulting character string as a related field name. However, when SYSID is specified as COMMON or when a subroutine is being processed the above information is lacking. In the case where SYSID=COMMON, the user may supply the missing information in the statement thus letting the language processor treat the audited field as a related field name.

ADFH817 INVALID NUMBER CONCATENATED TO SWITCH OR SQLWARN KEYWORD

Explanation: The SWITCH keyword must be followed by a digit from one to nine with no intervening blanks. The SQLWARN keyword must be followed by a digit from zero to seven with no intervening blanks.

ADFH818 ON OR OFF KEYWORD NOT FOUND

Explanation: A statement involving the testing or setting of an indicator could not detect the specified setting. One cause of this message could be the result of the item by item scan of each statement being out of synchronization because of a previous error.

ADFH819 ONLY THE STATEMENT SKSDISP = OFF OR STOP IS VALID

Explanation: Only SKSDISP = OFF OR STOP is valid.

ADFH820 INVALID STARTING LABEL CHARACTER

Explanation: The starting label specified in the STRTLBL statement must be a two-character string from the 35-character set A through Z and 1 through 9. Note that zero and blank are invalid.

ADFH821 PHASE OUT OF SEQUENCE

Explanation: For each new compilation the phases KEY, PRELIM, and PROCESS must appear in the above sequence. While all phases need not be present, those that are must be in the above order.

ADFH822 AUDIT DATA BASE LEG OUT OF SEQUENCE

Explanation: In the key phase if both P0 and P1 are present they must be in that order, P2 is invalid in this phase. In the PRELIM and PROCESS phases the required sequence is P0, P1, P2 though not all legs need be used.

ADFH823 TRXID MUST BE 2 DIGITS OR 2 CHARACTER LITERAL

Explanation: None

ADFH824 MODE MUST BE DIGIT BETWEEN 1 AND 6

Explanation: None

ADFH825 ONLY COUNTER1-9 AND COUNTERF AVAILABLE TO USER

Explanation: None

ADFH826 NUMERIC CONSTANT EXCEEDS 15 DIGITS

Explanation: The maximum number of digits which can be handled by the Auditor when performing arithmetic operations is 15 .

ADFH827 END OF FILE FOUND WHILE TRYING TO READ INPUT RECORD

Explanation: This error can occur when the scan has been thrown out of synchronization by a previous error. If this should occur on the final statement of the input stream, the language processor may attempt to read beyond this record while analyzing the statement. A second cause of this error occurs when an attempt is made to read an empty input file.

ADFH828 QUOTES AROUND SUBROUTINE NAME MUST ENCLOSE 16 POSITIONS

Explanation: The number of positions in a subroutine name as indicated by the positions between two single quotes (or an opening single quote and a delimiting comma) is not 16.

ADFH829 NO SUBROUTINE PARAMETER NAME ENTERED

Explanation: If SUBNAME is specified a PARMNAME must also be specified.

ADFH830 RELATED FIELD NAME LENGTH NOT BETWEEN 5 AND 8 CHARACTERS

Explanation: None

ADFH831 SUBROUTINE NAME MUST BE IN QUOTES

Explanation: None

ADFH832 ASTERISK IS INVALID CHARACTER UNLESS IN COLUMN 1

Explanation: An asterisk has been detected as the starting character of a sequence of one or more non-blank characters. This condition can occur when the scan of a statement has been thrown out of synchronization as the result of a previous error.

ADFH833 THE STARTING CHARACTER OF A STRING OF NON BLANK CHARACTERS IS UNIDENTIFIED

Explanation: As with error 832 this error can occur as the result of a previous error.

ADFH834 SKSDISP STATEMENT MAY ONLY BE USED IN P1 LEG OF KEY PHASE

Explanation: None

ADFH835 CONDITIONAL STATEMENTS IN THIS LEG NESTED MORE THAN 50 DEEP

Explanation: The number of outstanding IF and DO statements without any corresponding ENDIF and/or ENDDO statements is greater than the maximum limit of 50.

Note: Compiler generated IF statements which produce audit operation descriptors '16' and 'F2' are included in the nesting count. If KANAME=ALT is used , the generated 'F2' operation descriptor reduces the nesting limit to 48 .

ADFH836 MESSAGE NUMBERS IN THIS LEG EXCEED LIMIT OF 100

Explanation: The ERRORMSG, INFOMSG, MSG, WARNMSG and SETERROR statements all refer to a message number. The number of such statements in a given leg exceeds the maximum limit of 100.

ADFH837 BRANCH TARGETS IN THIS LEG EXCEED LIMIT OF 400

Explanation: IF, DO and ELSE statements all generate implicit branches to bypass specific sequences of statements depending upon the conditions detected. The number of targets of these implicit branches is limited to 400 in any leg. Each IF and ELSE needs one branch target while each DO requires two.

Note: Compiler generated IF statements which produce audit operation descriptors '16' and 'F2' are included in the branch count.

ADFH838 NUMBER OF GOTO LABEL NAMES IN THIS LEG EXCEEDS LIMIT OF 100

Explanation: The number of label names in any leg is limited to 100.

ADFH839 ONLY THE USE OF = OR EQ IS VALID FOR SETTING INDICATORS

Explanation: Only ON or OFF are valid.

ADFH840 THE CHANGED AND NULL INDICATORS CAN ONLY BE SET ON

Explanation: The procedural statements CHANGED and NULL can only set the corresponding indicator ON.

ADFH841 UNABLE TO IDENTIFY WHICH LEG WAS REQUESTED

Explanation: The language processor is unable to identify the leg (P0, P1, or P2) that was specified. This can occur as the result of a previous error or if the leg was not specified in the input.

ADFH842 UNABLE TO IDENTIFY WHICH PHASE WAS REQUESTED

Explanation: The language processor is unable to identify the phase (KEY, PRELIM or PROCESS) that was specified. As with error ADFH841 this can occur as the result of a previous error or if the phase was not specified in the input.

ADFH843 UNABLE TO OPEN SOURCE STATEMENTS INPUT FILE

Explanation: The language processor was unable to locate or open the file containing the input statements.

ADFH844 UNABLE TO IDENTIFY STATEMENT

Explanation: A valid operation code could not be generated from the Source statement. This could result from a scan error caused by a previous error.

ADFH845 AN ELSE NOT PRECEDED BY IF

Explanation: An ELSE was preceded by something other than an IF in the control structure. Each ELSE must be associated with a corresponding IF.

ADFH846 EXPECTED LITERAL OR NUMERIC CONSTANT NOT FOUND

Explanation: The expression requires a literal or numeric constant and it was not specified.

ADFH847 GOTO LABEL NAME PREVIOUSLY DEFINED

Explanation: Label name can only be used once.

ADFH848 INVALID USE OF KEYWORD

Explanation: Check statement syntax for proper use.

ADFH849 MISSING ENDIF, ENDDO OR ENDTWIN

Explanation: Missing ENDIF, ENDDO, or ENDTWIN. This error may be caused by previous syntax error.

ADFH850 EXPECTED LITERAL NOT FOUND

Explanation: Literal expected in statement. Check statement syntax.

ADFH851 EXPECTED KEYWORD NOT FOUND

Explanation: Compiler is expecting specific keyword. Check statement syntax.

ADFH852 EXPECTED RELATED FIELD NOT FOUND

Explanation: Related field name is expected in specific sequence. Check statement syntax. This error may be caused by previous syntax error.

ADFH853 LITERAL LENGTH ERROR

Explanation: Literal length is not valid for this statement or there may be a missing quote. Check statement syntax.

ADFH854 GOTO LABEL NAME EXCEEDS 8 CHARACTERS

Explanation: None.

ADFH855 EXPECTED NUMERIC CONSTANT NOT FOUND

Explanation: Numeric constant is expected in specific sequence. Check statement syntax. This error may be caused by previous syntax error.

ADFH856 NUMERIC CONSTANT VALUE ERROR

Explanation: Numeric constant is out of valid range for this statement. Check statement syntax.

ADFH857 ENDTWIN DOES NOT CLOSE DOTWIN STATEMENT

Explanation: An ENDTWIN statement is not preceded by a corresponding DOTWIN statement.

ADFH858 DATA DESCRIPTORS EXCEED LIMIT OF 50

Explanation: None.

ADFH859 QUOTES AROUND TBLROOT VALUE MUST ENCLOSE 16 POSITIONS

Explanation: None.

ADFH860 TBLNAME STATEMENT IS MISSING

Explanation: TBLNAME keyword is missing or may be misspelled.

ADFH861 TBLDESC STATEMENT IS MISSING

Explanation: TBLDESC keyword is missing or may be misspelled.

ADFH862 TABLE FIELD IS TOO LONG

Explanation: See Tables Specification section for table field lengths.

ADFH863 TABLE NAME OR DESCRIPTION IS NOT ENCLOSED IN QUOTES

Explanation: Table name or description must be enclosed in quotes.

ADFH864 FUNCTION IS NOT SPECIFIED FOR LAST TABLE ENTRY

Explanation: This error may result from a missing blank between argument and function values, a missing argument, or any other combination that may be interpreted as an odd number of literals by the compiler.

ADFH865 NOT CURRENTLY PROCESSING A TABLE--TBLNAME NOT ALLOWED

Explanation: See Tables Specification section for proper coding of tables under the TABLES phase or TBLROOT. This error may also be the result of an error in a previous table specification that results in an exit from tables processing in the compiler.

ADFH866 SQL WARNING CAN ONLY BE ' ' OR 'W'

Explanation: SQLWARNn can only be tested for ' ' or 'W'.

ADFH867 INVALID RELATIONAL OPERATOR WAS USED

Explanation: See statement syntax for allowed relational operators.

ADFH868 INVALID SPASQL VALUE

Explanation: SPASQL may be set to KSELECTn or n, where n is 3 to 9.

ADFH869 LITERAL HAS INVALID DBCS CHARACTER OR INCORRECTLY PAIRED SO/SI

Explanation: Check literal to make sure that valid double-byte codes are within SO/SI pair.

ADFH870 OUTLINE CODE WAS SPECIFIED INCORRECTLY

Explanation: Field outlining may be set to any subset or combination of 'OULR' with no embedded blanks or commas. It may also be set to 'N' to turn off outlining for the field.

ADFH871 INVALID SYNTAX FOR ARITHMETIC PROCEDURAL STATEMENT

Explanation: See statement syntax for procedural statements.

ADFH872 AUDITED FIELD CANNOT BE USED IN THE SETARRAY ASSIGNMENT

Explanation: The audited field cannot be the first field in an array.

ADFH873 ERROR OCCURRED IN STATIC AUDIT RULE SOURCE. SEE LISTING (DDNAME = MSGOUT)

Explanation: An error occurred in the creation of the static audit rule. See the listing data set that is specified above for the specific error messages.

ADFH874 DOTWIN LOOPS CANNOT BE NESTED

Explanation: A DOTWIN statement occurs within an existing DOTWIN loop. This is not allowed.

ADFS001 **OPEN ERROR ON DATA SET WHERE DDNAME = OUTDATA**

Explanation: None

ADFS002 **OPEN ERROR ON OUTPUT DATA SET WHERE DDNAME = STADOUT**

Explanation: None

ADFS003 **OPEN ERROR ON DATA SET WHERE DDNAME = KEYAUDT**

Explanation: None

ADFS004 **DUPLICATE AUDIT ROUTINE FOUND FOR (last 8 characters of key printed)**

Explanation: A duplicate audit routine consists of a duplicate audit group code and eight-character field name.

ADFS005 **AUDITS WITHIN (8 character audit group code printed) NOT CONTIGUOUS IN SOURCE. ONLY 1 PROCESSED**

Explanation: An audit routine was found that was entered after the audit group code that it belongs to, had been ended. The audit routine that is not contiguous will not be processed.

ADFS006 **EXCEEDED MAXIMUM NUMBER OF AUDIT ROUTINES WITHIN AN AUDIT GROUP CODE**

Explanation: No more than 6000 audit routines per audit group code are allowed.

ADFS007 **EXCEEDED MAXIMUM NUMBER OF 8 CHARACTER AUDIT GROUP CODES**

Explanation: No more than 100 different audit group codes can be entered in a single run.

ADFS008 **EMPTY DATA SET FOR INPUT**

Explanation: None

INDEX

Special Characters

A

B

C

D

E

H

I

N

O

P

Q

R

S

T

U

V

W

X

Z

IMS Application Development Facility II Version 2 Release 2
Application Development Reference
SH20-6594-01

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.)

SH20-6594-01

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 Reference
SH20-6594-01

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.)

SH20-6594-01

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 Reference
SH20-6594-01

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.)

SH20-6594-01

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 Reference
SH20-6594-01

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.)

SH20-6594-01

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®

Publication Number
SH20-6594-01

File Number
S/370/4300-32

Program Number
5665-348

IBM®

SH20-6594-01