IMS DL/I File Access Guide

Release 8.1.3

November 2013

METASUITE

IKAN Solutions N.V.
Kardinaal Mercierplein 2
B-2800 Mechelen
BELGIUM

Table of Contents

CHAPTER 1

About This Manual

The MetaSuite file access for IMS describes the use of MetaSuite with IMS database files. In the VSE/AF operating system environment, users commonly refer to the IMS system as "DL/I". Since the IMS and DL/I systems differ only (with minor exceptions) in the Operating System in which they execute, we will refer only to "IMS" in this supplement.

This supplement is intended for users with some experience with MetaSuite. It provides the information you need to use the MetaSuite IMS Database File Access, including discussions on IMS concepts, defining an IMS database for use with MetaSuite and using MetaSuite commands to access IMS databases.

Because most MetaSuite commands are independent of the environment in which MetaSuite operates, only those commands that pertain to IMS database definition and access are described in this supplement. The MetaSuite User and Reference Guides are the primary sources of information about MetaSuite.

1.1. Prerequisites

Readers are expected to be familiar with IMS.

1.2. Related Publications

The MetaSuite User and Reference Guides describe the different MetaSuite components and provide examples for using MetaSuite. Those guides should be available for reference during the installation and test procedures described here.

The following table gives an overview of the complete MetaSuite documentation set.

Release Information	Release Notes 8.1.3
Installation Guides	• BS2000/OSD Runtime Component • DOS/VSE Runtime Component • Fujitsu Windows Runtime Component • MicroFocus Windows Runtime Component • MicroFocus UNIX Runtime Component • OS/390 and Z/OS Runtime Component • OS/400 Runtime Component • VisualAge Windows Runtime Component • VisualAge UNIX Runtime Component • VMS Runtime Component

User Guides	• INI Manager User Guide • Installation and Setup Guide • Introduction Guide • MetaStore Manager User Guide • MetaMap Manager User Guide • Generator Manager User Guide
Technical Guides	• ADABAS File Access Guide • IDMS File Access Guide • IMS DLI File Access Guide • RDBMS File Access Guide • XML File Access Guide • Runtime Modules • User-defined Functions User Guide

If you are unfamiliar with MetaSuite, the following technical description provides you with a brief overview.

The MetaSuite System	MetaSuite is designed for data retrieval, extraction, conversion and reporting. It includes a workstation-based graphical user interface and a mainframe runtime component.
MetaSuite Database Interfaces	MetaSuite can access data from a number of database management systems, using the same commands, program structure and retrieval techniques used for non-database files. Each database interface is available as an optional enhancement to the base product.
MetaMap Manager	MetaMap Manager is the MetaSuite tool used to define models. Such models are intuitively built by describing overall program specifications, input file definitions (data and process) and target file definitions (data and process).
MetaStore Manager	MetaStore Manager is a tool that provides metadata maintenance and documentation services.
Generator Manager	The Generator Manager is the system administration tool.All kinds of basic functionalities and customization possibilities are supported by this tool.

CHAPTER 2
Introduction

Because most MetaSuite commands are independent of the environment in which MetaSuite operates, only those commands that pertain to an IMS database definition and access are described in this supplement.

Additional chapters in this guide include:

- IMS (DL/I) Database Management System (page 4)
- Defining an IMS Database to MetaSuite (page 6)
- Programming With File Access for IMS (page 16)

CHAPTER 3

IMS (DL/I) Database Management System

IMS is the database management system (DBMS) distributed by the IBM Corporation for the z/OS operating system environment. It supports the MetaSuite database concepts of field, record, link, and index, although the link and index concepts are mostly hidden from the view of the application programmer. There is no concept in IMS, which corresponds precisely to the MetaSuite file concept, so each separate IMS application view of a database ("Program Control Block" or "PCB" in IMS jargon) is treated as a single MetaSuite file.

This chapter presents an overview of the IMS concepts and terms you should know before using MetaSuite to access an IMS.

3.1. MetaSuite and IMS Terminology

All MetaSuite File Access components use the same terminology when referring to data structures. The MetaSuite terminology, though, may differ from that of an individual DBMS. The relationships between the basic MetaSuite and IMS terms are as follows:

MetaSuite and IMS Terms

MetaSuite	IMS
File	IMS PCB
Record	IMS Segment
Field	Field
Index	IMS Index

The MetaSuite terms are described separately below, along with their IMS correspondences.

Segment

The basic unit of data storage in an IMS database is an IMS segment. A segment is a collection of related data items, or fields. There can be many segments in a database, each with its own definition.

In MetaSuite file access for IMS:

- An IMS segment is defined and referred to as a RECORD.

Parent-Child Relationship

The IMS parent-child relationship describes a single relationship between two specific segment types. There are two important restrictions to a parent-child relationship, in comparison with a common relationship:

- IMS, at any one time, allows only a single relationship between two specific segment types to be used.
- A parent-child relationship can be traversed only in the parent-to-child direction. Thus, a reference to a child segment in an application program unambiguously identifies both the segment type and the relationship type.

Consequently, parent-child relationships cannot be specified in the application program, so they are neither defined in the MetaStore.

DataBase Definition (DBD)

The Data Base Definition (DBD) is a data module describing an entire IMS database to the DBMS. It includes descriptions of all the entities stored in the database, specifications of the physical datasets used to hold the various entities, and the access methods that will be used by the DBMS. The process of generating a DBD is referred to as a DBDGEN.

The DBDGEN listing will be a useful source of information when defining IMS databases to the MetaStore.

Program Specification Block (PSB)

An IMS Program Specification Block (PSB) is a data module that serves a particular application, and defines which portions of the physical database can be used by that program, and how the program will view the data.

The PSB-name must be specified in the JCL used to execute MetaSuite-generated IMS programs. The process of generating a PSB is referred to as a PSBGEN. The listing created by a PSBGEN, will be a useful source of information when defining IMS databases to the MetaStore.

Program Control Block (PCB)

Each PSB consists of one or more Program Control Blocks (PCBs). Each PCB presents a logical view of the database, and determines which segments and relationships are available for use.

As noted above, there are restrictions on the actual combinations of segments and parent-child relationships that may be accessed through a specific PCB:

- there may be only one parent-child relationship between any two segments in the PCB
- they form a strict hierarchy from the first parent segment (called the "root" segment) to all dependent segment types

In MetaSuite file access for IMS:

- An IMS PCB is defined and referred to as a FILE.

Index

Each IMS index is actually an IMS database, called an "index database". The segments in an index database are not accessed by a MetaSuite application, but are used instead by the DBMS to control the sequence in which the other database segments are accessed. The index database is "invisible" to the application program. There are, however, two important restrictions on the use of IMS indexes:

- Only a single index may be used in any given PCB
- An index must be based on the root segment of the PCB

In MetaSuite file access for IMS:

- An IMS index is defined and referred to as an INDEX.

CHAPTER 4

Defining an IMS Database to MetaSuite

4.1. Overview

This chapter describes how to provide MetaSuite with access to the definitions that it needs to process data in an IMS database.

Note that the descriptions in this chapter use both the MetaSuite and IMS terminology for data entities, as appropriate. The correspondences between the MetaSuite and IMS terminology are discussed in detail in IMS (DL/I) Database Management System (page 4). These correspondences are summarized in the following table.

MetaSuite	IMS
File	IMS PCB
Record	IMS Segment
Field	Field
Index	IMS Index

Before you begin to define database or non-database files to the MetaStore, you must locate the appropriate sources of information about those files. In the case of IMS, the source documents which provide the information necessary to define files, records, indexes, and fields to MetaStore are the DBDGEN and PSBGEN job source listings. There will be several different PSBGEN jobs, because each PSB has been set up for a specific application. Refer only to the PSB, which you will be using with your MetaSuite generated program.

The definition of the MetaSuite IMS file, records and fields can be automated when you have a COBOL Copy Book or PL/I Include Book that describes the segment layout of those segments that you want to group in one file. This COBOL Copy Book or PL/I Include Book can be used by the *Collect File* option in the MetaStore Manager to create the proper base definition of the IMS segments in MDL format. Refer to the *MetaStore Manager User Guide* for more information.

4.2. Commands

The following table lists the MetaSuite Commands used to manually code IMS data definitions.

Command	Used to define
ADD FILE	An IMS PCB
ADD RECORD	An IMS segment
ADD FIELD	An IMS field
ADD INDEX	An IMS index

Each command is described separately below.

4.3. FILE

The ADD FILE command defines an IMS file to the MetaStore. The general syntax for the ADD FILE command is described in the *MetaMap Manager User Guide*. The options that refer to IMS are described below.

The ADD FILE command will describe all segments within an IMS PCB.

Format

```
ADD FILE File-name
    [VERSION File-version]
    TYPE IMS
      'PCB-name'
    [RULE Business-rule]
```

Usage

The ADD FILE command defines an IMS file to the MetaStore.

You can find the information you need to code on the ADD FILE command in the PSBGEN which will be used when accessing the file in a MetaSuite generated program.

Each PCB defined in a PSBGEN job will be identified by the following statement:

```
PCBname PCB TYPE=DB,DBDNAME=DBname,KEYLEN=len
```

PCBname may not be specified in the PSBGEN, or it may be spelled the same as a PCB name in another PSBGEN. In either of those cases, you will need to use a different name for *file-name*. Note that you must make sure that the DBDGEN job, which you are using, defines the same DBD as specified in the *DBDname* parameter of the PCB statement in the PSBGEN.

The name of the PSB in the PSBGEN job can be found at the end of the job by the indication:

```
PSBNAME=PSB-name
```

You can use the *Collect File* option in the MetaStore Manager to generate the IMS file definition from a COBOL Copy Book or a PL/I Include Book.

File-name

Required. *File-name* is an arbitrary name of up to 32 characters, although it is suggested that the PCB name be used. It can include alphabetic characters, numbers, the embedded characters #, @, $, embedded hyphens and embedded underscores. It must begin with an alphabetic character.

File-version

Optional. The *VERSION* option specifies the version-number of a file.

PCB-name

Required. *PCB-name* is the name of the PCB as found in the PSBGEN.

Business-rule

Optional. The *RULE* option is used to add a business rule documenting your file.

Example 1: IMS Database Definition

An IMS customer database is to be defined to the MetaStore. After examining the DBDGEN and PSBGEN listings for this database, it is determined that the PSB that we are interested in is named "PSB1". All of the PCBs from this PSB will be defined as MetaSuite files. The applicable statements from our PSBGEN listing are as follows:

```
DEMOPCB1 PCB TYPE=DB,DBDNAME=DEMODBD1,KEYLEN=17
.
DEMOPCB2 PCB TYPE=DB,DBDNAME=DEMODBD1,KEYLEN=27,
    PROCSEQ=DEMOINDX
.
PSBGEN LANG=COBOL,PSBNAME=PSB1,CMPAT=NO
```

Using the PSBGEN statement shown above, the MetaSuite ADD FILE commands required for our example database would be as follows:

```
ADD FILE DEMOPCB1 TYPE IMS   'DEMOPCB1'
ADD FILE DEMOPCB2 TYPE IMS DBNAME 'DEMOPCB2'
```

Example 2: IMS Database Definition

The applicable statements from our PSBGEN listing are as follows:

```
DEMOPCB1 PCB TYPE=DB,DBDNAME=DEMODBD1,KEYLEN=17
.
DEMOPCB2 PCB TYPE=DB,DBDNAME=DEMODBD1,KEYLEN=27,
    PROCSEQ=DEMOINDX
.
PSBGEN LANG=COBOL,PSBNAME=PSB1,CMPAT=YES
```

Using the PSBGEN statement shown above, the MetaSuite ADD FILE commands required for our example database would be as follows:

```
ADD FILE DEMOPCB1 TYPE IMS DBNAME 'DEMOPCB1'
ADD FILE DEMOPCB2 TYPE IMS DBNAME 'DEMOPCB2'
```

4.4. RECORD

IMS database segments are defined to the MetaStore using the ADD RECORD command. The general syntax for the ADD RECORD command is described in the *MetaMap Manager User Guide*. The syntax of the ADD RECORD command when used to define IMS database segments includes additional options necessary for database processing, and is described below.

Format

```
ADD RECORD Record-name [OF File-name]
    SIZE maximum-record-size
    [STORAGE-KEY storage-keyfield]
    DBNAME 'IMS-name'
    [RULE Business-rule]
```

Usage

The ADD RECORD command defines an IMS database segment to the MetaStore.

You can find the information you need to code on the ADD RECORD command in both the DBDGEN, and the PSBGEN.

The following illustrates the format of the DBDGEN segment definition statement:

```
SEGM NAME=segname,PARENT=parent,BYTES=size
    FIELD NAME=(Field-name,SEQ,U),BYTES=fieldsize
```

The following illustrates the format of the PSBGEN segment definition statement:

```
SENSEG NAME=segname,PARENT=parent
```

You can use *Collect File* option in the MetaStore Manager to generate the base IMS record definition from a COBOL Copy Book or a PL/I Include Book. You will still need to compare the following characteristics within the MetaSuite MetaStore with the information in the DBDGEN and the PSBGEN:

- Add the DBNAME to the record
- Compare the record size with the size on the DBDGEN. You should modify the record size to the segment size on the DBDGEN if different.
- Add the STORAGE-KEY to the record

Record-name

Required. *Record-name* is an arbitrary name of up to 32 characters. You can use the name of the segment as specified in the DBD and PSBGENs.

File-name

Optional. *File-name* is the name of the file to which the record belongs. If this option is omitted, the record is defined within the current file; that is, within the file named on the most recent ADD FILE statement in the command stream.

Maximum-record-size

Required. *Maximum-record-size* is the record size. The record size should simply be specified as the value of the BYTES parameter of the SEGM statement in the DBDGEN job. The number specified for the record size need not be exact, but must be at least as large as the actual record size.

Specifying a record size, which is too small for the actual record will cause unpredictable processing results.

Storage-keyfield

Optional. *Storage-keyfield* is the name of the field, by which you can randomly obtain the record. This option must be specified for any record whose DBDGEN definition includes a field with the "SEQ" attribute. You can determine if this is the case by examining the DBDGEN statements for the record being defined. If one of the FIELD statements, following the SEGM statement for the record in question, appears as follows:

```
FIELD NAME=(IMS-field,SEQ,U),START=position
```

, then this is the storage key for the record.

At this point, you must choose a MetaSuite field-name for this field, and specify that name as the *storage-keyfield* of the STORAGE-KEY option entered on the ADD RECORD command. Later, you will define that field within the MetaSuite record using the same field-name you have selected, and the value shown for *IMS-field* in the DBDGEN statement must be specified as the internal database name (DBNAME) for that field. (See "Field" later.)

IMS-Name

Required. *IMS-Name* must be set to the segment name of the record, as specified in the DBDGEN and PSBGEN statements.

Business-rule

Optional. The *RULE* option is used to add a business rule documenting your record.

Example

To continue our customer database example, assume that we want to add four record definitions to the MetaSuite MetaStore for the PCBs DEMOPCB1 and DEMOPCB2. The first step would be to examine the DBDGEN and PSBGEN jobs. The applicable statements from the DBDGEN job would appear as follows:

```
SEGM NAME=CUSTOMER,PARENT=0,BYTES=104
    FIELD NAME=(CUSTNUM,SEQ,U),BYTES=10,...
.
SEGM NAME=INVOICE,PARENT=CUSTOMER,BYTES=40
FIELD NAME=(ORDNUM,SEQ,U),BYTES=7,...
.
SEGM NAME=OREMARK,PARENT=INVOICE,BYTES=72
.
SEGM NAME=ITEM,PARENT=INVOICE,BYTES=3226
```

The applicable statements from the PSBGEN job would appear as follows:

```
DEMOPCB1  PCB TYPE=DB,DBDNAME=DEMODBD1,KEYLEN=17
   SENSEG NAME=CUSTOMER,PARENT=0
          INDICES=DEMOINDX,PROCOPT=G
   SENSEG NAME=INVOICE,PARENT=CUSTOMER,PROCOPT=G
   SENSEG NAME=OREMARK,PARENT=INVOICE,PROCOPT=G
   SENSEG NAME=ITEM,PARENT=INVOICE,PROCOPT=G
DEMOPCB2  PCB TYPE=DB,DBDNAME=DEMODBD1,KEYLEN=27
          PROCSEQ=DEMOINDX
   SENSEG NAME=CUSTOMER,PARENT=0,PROCOPT=G
    SENSEG NAME=INVOICE,PARENT=CUSTOMER,PROCOPT=G
```

Using the DBDGEN and PSBGEN statements shown above, the ADD RECORD commands would be coded as follows:

```
ADD RECORD CUST-PCB1 OF DEMOPCB1 SIZE 104 STORAGE-KEY CUST-NUMBER DBNAME 'CUSTOMER'
ADD RECORD CUST-PCB2 OF DEMOPCB2 SIZE 104 STORAGE-KEY CUST-NUMBER DBNAME 'CUSTOMER'
ADD RECORD INV-PCB1 OF DEMOPCB1 SIZE 40 STORAGE-KEY ORDER-NUMBER DBNAME 'INVOICE'
ADD RECORD INV-PCB2 OF DEMOPCB2 SIZE 40 STORAGE-KEY ORDER-NUMBER DBNAME 'INVOICE'
ADD RECORD OREMARK OF DEMOPCB1 SIZE 72 DBNAME 'OREMARK'
ADD RECORD ITEM OF DEMOPCB1 SIZE 3226 DBNAME 'ITEM'
```

The IMS-names and sizes are determined from the DBDGEN statements. The breakdown of which records are available through which PCBs is taken from the PSB statements. Note that we have chosen the MetaSuite names CUST-NUMBER and ORDER-NUMBER for the IMS fields CUSTNUM and ORDNUM, respectively. The IMS field names will be specified as the internal database names when we define these MetaSuite fields later. (See "Field" later.)

4.5. INDEX

You should define MetaSuite indexes:

- Whenever the database is an indexed database (HISAM, HIDAM, etc.),
- Whenever the database has secondary indexes defined.

The ADD INDEX command is used to define those indexes. The syntax and usage of this command is described below.

Format

```
ADD INDEX index-set-name BASED ON index-field-name
```

Usage

The ADD INDEX command defines an IMS primary or secondary index to the MetaSuite MetaStore. You can find the information you need to code on the ADD INDEX command in the DBDGEN.

The following illustrates the format of the DBDGEN statement:

```
DBD NAME=DBDname,ACCESS=HISAM
SEGM NAME= segname,PARENT=parent,BYTES=size
FIELD NAME=(Field-name,SEQ,U),...
FIELD NAME=Field-name,...
.
LCHILD NAME=(index-DBDname1),PTR=INDX
  XDFLD NAME=index-name,SRCH=Field-name
```

An IMS index always needs to be added manually to the MetaSuite MetaStore. Its definition can not be generated by the *Collect File* option in the MetaStore Manager.

Index-set-name

Required. *Index-set-name* is the name by which MetaSuite users would refer to the index. It may be any unique name, although it is suggested that the name of the index database (index-DBDname) be used.

Index-field-name

Required. *Index-field-name* is the MetaSuite field-name of the field from which the indexed values are drawn. The IMS field names will be specified as the internal database names when we define these MetaSuite fields later. (See "Field" later.)

Example

In our DBDGEN job, the sample database is defined as follows:

```
DBD NAME=DEMODBD1,ACCESS=HISAM
SEGM NAME=CUSTOMER,PARENT=0,BYTES=104
FIELD NAME=(CUSTNUM,SEQ,U),...
FIELD NAME=CUSTNAME,...
.
LCHILD NAME=(CUSTINDX,CNAMINDX),PTR=INDX
  XDFLD NAME=CUSTNAMX,SRCH=CUSTNAME
```

In addition, an index database is defined as follows:

```
DBD NAME=CNAMINDX,ACCESS=INDEX
DATASET ...
SEGM NAME=CNAMXSEG,BYTES=...
FIELD NAME=(CNAMIX1,SEQ,U),...
LCHILD NAME=(CUSTOMER,DEMODBD1),INDEX=CUSTNAMX
DBDGEN
FINISH
```

Finally, the definitions of the PCBs in our PSBGEN listing contain the following statements:

```
DEMOPCB1 PCB TYPE=DB,DBDNAME=DEMODBD1,KEYLEN=17
.
DEMOPCB2 PCB TYPE=DB,DBDNAME=DEMODBD1,KEYLEN=27,
       PROCSEQ=CNAMINDX,PROCOPT=G
```

By inspecting the main database DBDGEN statements, we see that two indexes may be defined. Since the DBD ACCESS statement specifies HISAM, we will define an index (the "primary index") based on the key field of the root segment (CUSTNUM in the CUSTOMER segment). Because the DBDGEN also contains an XDFLD statement, we know that there is a secondary index. The segment accessed by the index (called the "target segment" in IMS jargon) is also the CUSTOMER segment, and the index-field through which the index is accessed is the XDFLD (CUSTNAMX). The definitions for these indexes are as follows:

```
ADD INDEX CNUMINDX BASED ON CUST-NUMBER
ADD INDEX CNAMINDX BASED ON CUST-NAME
```

Note that CUST-NUMBER and CUST-NAME are MetaSuite field-names.

When these fields are defined, they will be defined with the DBNAME option. The DBNAME for CUST-NUMBER must be the name shown on the FIELD statement in the DBDGEN (CUSTNUM). The DBNAME for CUST-NAME must be the name shown on the XDFLD statement in the DBDGEN (CUSTNAME). Note also that the ADD FIELD commands to define these fields must precede the ADD INDEX commands when they are imported in the MetaStore Manager. Once these indexes are defined to the MetaSuite MetaStore, then the full range of index-based functions (index-sequential processing, START processing, etc.) may be used by the MetaSuite application programs.

4.6. FIELD

The ADD FIELD command is used to define all database fields. The general syntax is described in the *MetaMap Manager User Guide*. Only IMS specific characteristics are described in this supplement.

Format

```
ADD FIELD field-name [OF { record | group-field }]
    [POSITION start-value]
    [SIZE characters]
    [OCCURS number-times
          [DEPENDING ON depend-field]]
    [TYPE { CHARACTER |
          BIT number |
          FLOAT |
          BINARY [DECIMAL places] |
          PACKED [DECIMAL places] [UNSIGNED] |
          ZONED [DECIMAL places]
          [UNSIGNED | [SEPARATE] LEADING]] } ]
    [DATE 'format']
    [EDIT 'mask']
    [INITIAL value]
    [LIMITS (minimum TO maximum)]
    [DBNAME 'IMS-name']
    [RULE Business-rule]
```

Usage

The ADD FIELD command defines an IMS segment field.

The syntax is identical to that described in the *MetaMap Manager User Guide*, with one exception: every field which is named as the STORAGE-KEY of a database record, and every field which is named as the BASED ON field of a database index, must be defined with the DBNAME option.

The discussion of the ADD FIELD command which follows includes only those options for which special considerations apply when defining IMS fields to the MetaSuite MetaStore. Note that many fields available in an IMS record will not be defined in the DBDGEN job. Typically, a few large group-fields will be defined with FIELD statements in the DBDGEN job, and the fine structure (the individual fields) will be defined only in the application programs. Consequently, you will need to consult with your systems staff to locate descriptions of most of the fields available in your database - the DBDGEN and PSBGEN jobs will provide only descriptions of the key fields and the large group-fields.

Note also, that various kinds of IMS "field-level sensitivity" may impact the appearance of the records returned by the DBMS. If the PSBGEN job contains SENSEG statements which specify PROCOPT=K, then only the key fields of those records may be defined to the MetaSuite MetaStore. Likewise, if the PSBGEN job contains SENSEG statements which are followed by SENFLD statements, then only the fields defined by the SENFLD statements may be defined to the MetaSuite MetaStore. The following discussions assume that no PSBGEN SENSEG...PROCOPT=K or SENFLD statements are present, and therefore, the DBDGEN definitions of the IMS fields may be used.

- You can use the *Collect File* option in the MetaStore Manager to generate the base IMS field definition from a COBOL Copy Book or a PL/I Include Book. You will still need to add the DBNAME to the fields that are used as storage-keys on index fields within the MetaSuite MetaStore.

Start-value

If the field is defined in the DBDGEN job, then the value of the START parameter on the FIELD statement in that job is used for *start-value*, and the *record* would be specified (OF *record*). If the field is not defined in the DBDGEN job (i.e., it is a subfield defined only in the application programs which access the database), then you must use the normal rules shown in the *MetaMap Manager User Guide* to determine field position (OF *group-field*).

Characters

If the field is defined in the DBDGEN job, the value of the BYTES parameter in the FIELD statement in that job is used for *characters*. If the field is not defined in the DBDGEN job, you must use the normal rules to determine field size.

TYPE field-type

If the field is defined in the DBDGEN job, *field-type* may be determined from the DBDGEN TYPE option of the FIELD statement as follows:

This DBDGEN Type	Is Equivalent to This MetaSuite Type . . .
X	Could be any type.
P	PACKED
C	CHARACTER or ZONED
F	BINARY
H	BINARY

If the field is not defined in the DBDGEN job, you must use the normal rules for determining field type.

IMS-name

The DBNAME parameter is required only for fields, which will be defined either as record STORAGE-KEY fields or as index BASED ON fields. When the field is defined as a STORAGE-KEY or when it is defined as the BASED ON field of the primary index for the database, *IMS-name* must be the same as the value of the NAME parameter on the FIELD statement in the DBDGEN which defines the field. In the case of secondary indexes, the field is known to IMS by two names contained in the DBDGEN: once on a FIELD statement and once on an XDFLD statement. You will use the FIELD statement to obtain all the information about a secondary index field except the DBNAME. For a secondary index field, the *IMS-name* you specify must be taken from the XDFLD statement.

Example

Continuing the definition of our example DEMODBD1 database, we will show the field definitions for all of the STORAGE-KEY fields and INDEX base-fields, plus a few others. The applicable fields from the DBDGEN job would be as follows:

```
SEGM NAME=CUSTOMER,PARENT=0,BYTES=104
FIELD NAME=(CUSTNUM,SEQ,U),BYTES=10,START=1,TYPE=X
FIELD NAME=CUSTNAME,BYTES=20,START=11,TYPE=X
.
LCHILD NAME=(CNAMINDX),PTR=INDX
```

```
XDFLD NAME=CUSTNAMX,SRCH=CUSTNAME
.
SEGM NAME=INVOICE,PARENT=CUSTOMER,BYTES=40
FIELD NAME=(ORDNUM,SEQ,U),BYTES=7,START=1,TYPE=P
FIELD NAME=ORDCPON,BYTES=10,START=8,TYPE=X
.
SEGM NAME=ORMARK,PARENT=INVOICE,BYTES=72
FIELD NAME=OREMSEQ,BYTES=2,START=1,TYPE=H
FIELD NAME=OREMTEXT,BYTES=70,START=3,TYPE=C
.
```

Using the DBDGEN statement shown above, the MetaSuite ADD FIELD commands would be coded as follows:

```
ADD FIELD CUST-NUMBER POSITION 1 SIZE 10 DBNAME 'CUSTNUM'
ADD FIELD ORD-NUMBER POSITION 1 SIZE 7 TYPE PACKED DBNAME 'ORDNUM'
ADD FIELD ORD-CUST-PO-NUMB POSITION 8 SIZE 10 TYPE MIXED
ADD FIELD REMARK-SEQ POSITION 1 SIZE 2 TYPE BINARY
ADD FIELD REMARK-TEXT POSITION 3 SIZE 70 TYPE CHARACTER
```

Note the definition of DBNAME values for the various storage-key and index base-fields.

CHAPTER 5

Programming With File Access for IMS

5.1. Overview

This chapter describes how to use the MetaMap Manager commands that access information stored in an IMS database.

- Data source commands define the SourceFile, ExternalArray and GlobalFieldobjects to be used during the program processing. For IMS SourceFiles,the SourceFilePath can specify how the IMS records need to be accessedwithin the SourceFile.
- TargetFile objects define the output you want to generate.
- Procedural commands define the processing you want to occur, if any.

These program sections are described in detail in the MetaSuite User and Reference Guides.

The SourceFile objects may differ in use with an IMS SourceFile than with a non-database SourceFile. Target objects are unaffected by access to an IMS SourceFile.

Note that the descriptions in this chapter use the MetaSuite terminology exclusively. The correspondences between the MetaSuite and the IMS terminology are discussed in detail in Defining an IMS Database to MetaSuite (page 6). These correspondences are summarized in the following table.

MetaSuite Commands that Define IMS Data Structures

Command	Used to define
ADD FILE	An IMS PCB
ADD RECORD	An IMS segment
ADD FIELD	An IMS field
ADD INDEX	An IMS index

5.2. Programming Considerations

When coding a program that accesses an IMS database, you should be aware of the considerations below.

Accessing the database

In general, you access an IMS database as you would access a non-database SourceFile.

You define a PCB within a PSB through a SourceFile and a SourceFilePath object, whose options define whether the database is to be accessed automatically by MetaSuite or through your procedural code.

Processing Sequence

You must be aware of the processing sequence of a MetaSuite program, to avoid issuing a database command when no successful access is possible. For example, assume you try to access the IMS database from a SourceFile initial procedure for another SourceFile. The access request will be unsuccessful, unless you have already specified the SOURCEFILE command for the database, because the database has not yet been opened. See the "Order of Execution" topic in the *MetaMap Manager User Guide* for information about program processing sequence.

Navigating the Database

There are many ways to access an IMS database, some more efficient than others. If efficiency is a consideration, consult your systems staff for assistance.

Program commands

The MetaMap Manager commands and most procedural commands are unaffected by the use of IMS database SourceFiles. Refer to the *MetaMap Manager User Guide* for the syntax of these commands.

The following MetaSuite commands differ in their use with IMS databases, they are discussed in this chapter:

```
SOURCEFILE     EXCLUDE
GET            START
```

5.3. SourceFile

The options of the SourceFile objects are different for automatic, controlled and controlled by SourceFiles:

Automatic SourceFile object:

```
SOURCEFILE SourceFile-name [PREFIX 'prefix']
PATH (entry-record [VIA index-name]
      [{,subordinate-record VIA related-record
      [OCCURS number times]} …])
[MATCH (match-key,…) ]
```

Controlled SourceFile object:

```
SOURCEFILE SourceFile-name [PREFIX 'prefix']
CONTROLLED
PATH (entry-record [VIA index-name]
      [{,subordinate-record VIA related-record
      [OCCURS number times]} …])
```

Controlled By SourceFile object:

```
SOURCEFILE SourceFile-name [PREFIX 'prefix']
CONTROLLED BY controlling-SourceFile KEY = key-field
PATH (entry-record [VIA index-name]
      [{,subordinate-record VIA related-record
      [OCCURS number times]} …])
```

Each option is described separately on the following pages.

SourceFile-name

Names a *SourceFile* that has been defined to the MetaSuite MetaStore. Remember that a MetaSuite IMS file is usually assigned the name of an IMS PCB.

Prefix

Prefix is exactly four characters, including alphabetic characters, numbers, and embedded hyphens, beginning with an alphabetic character.

The PREFIX option allows the same definitions to be used in multiple SourceFile objects. Note that each reference to an object within the SourceFile will be prefixed with the *Prefix*.

PATH

```
PATH (entry-record [VIA index-name]
    [{,subordinate-record VIA related-record
    [OCCURS number-times]}...])
```

The PATH option is part of the SOURCEFILE command. It is mandatory for all IMS SourceFiles. The PATH option is used to simplify the processing of a database by allowing the user to view data from multiple record types as a single unit of data. This process of collecting data from multiple records to be treated as a single unit of data is sometimes called "file flattening". The PATH option identifies the record types to be processed, and expresses the relationships between the records in hierarchical terms. The records named in the path specification are said to be either "path records" or "associated records".

Path Records

A "path" is a continuous hierarchical route through the records of the database. For example, we might define a path through our example database as beginning at the CUSTOMER record, proceeding to the INVOICE record, and finally to the ITEM record. The system would process this path by obtaining the first CUSTOMER record in the database, the first INVOICE record for that CUSTOMER, and the first ITEM record for that INVOICE. This collection of data would then be available to the MetaSuite program procedures as the first path of data. The system would then attempt to obtain another ITEM record for the same INVOICE and, if successful, would return the new ITEM record along with the old INVOICE and CUSTOMER records as the next path of data. This process continues until there are no more ITEM records for the first INVOICE, at which point the system will obtain the next INVOICE record for the first CUSTOMER along with its first ITEM record.

Similarly, when there are no more INVOICE records for the first CUSTOMER, the next CUSTOMER record will be obtained and the processing of its INVOICE and ITEM records will continue as described above. Note that each successive record named in the MetaSuite path must be a "dependent" of the record named previously in the path. The parent/child relationships used in determining dependency are those specified in the PSBGEN job for the PCB in question. A given PCB may specify a different set of parent/child relationships than another given PCB, so it is important to be aware of the definition of the actual PCB in use.

Associated Records

It is sometimes necessary to process records which are related to a path record, but which do not directly participate in the hierarchy. Using the same CUSTOMER, INVOICE, ITEM path described above, assume that we would like to access information from the first five invoice remarks (INV-REMARK) records for each order. The INV-REMARK record would be defined in the MetaSuite path specification as an "associated record" of the INVOICE record. The system would process the main CUSTOMER, INVOICE, ITEM path as described above, but each time a new INVOICE record was obtained, up to five INV-REMARK records associated with that INVOICE record would also be obtained. Note that the user has the option of requesting either a single occurrence or multiple occurrences of an associated record in the path. Again, each associated record in a path must be the dependent of the previous path record. In summary, all associated record types are related to path records, but do not participate in a hierarchical path.

Identifying the Entry Record and Its Access Technique

`Entry-record [VIA index-name]`

Required. *Entry-record* is the highest level record in the path hierarchy, the record at which the navigation through the database begins.

Without the VIA option, the entry-record can be any record defined in the SourceFile. Traversal through the database will be based on this record type.

With the VIA option, certain conditions must apply:

The *Entry-record* must be the "root segment" as defined in the PCB in use.

- The PCB must have been defined in the PSBGEN job with the PROCSEQ parameter, naming an index database as the processing sequence. Index-name must be the name of the index specified in the PROCSEQ parameter.
- When the above two conditions are true, the entry records will be retrieved in index sequence. The VIA option is needed only if you intend to use the START or GET commands within your MetaSuite program.

Identifying Subordinate Records

`subordinate-record VIA related-record`

Optional. *Subordinate-record* names a dependent record in the database, as defined by the PCB in use. Because each PCB may define a different hierarchy in the database, it is important to check the PCB definition in the PSBGEN job to determine which records are dependents of each other. Up to 15 subordinate records may be specified following the entry record.

The *VIA option* explicitly names the relationship between the subordinate record and some other previously named record in the path specification. *Related-record* names a previous record in the path.

Identifying Associated Records

`OCCURS number-times`

The OCCURS option is used to indicate that a record is to be an associated record. *Number-times* may be in the range of 1 to 32,767 that indicates the number of occurrences of the record you want to retrieve.

Example 1: Single path

```
SOURCEFILE DEMOPCB1
PATH (CUSTOMER, INVOICE VIA CUSTOMER)
```

The SourceFile will retrieve CUSTOMER records from the database in sequential order. Each CUSTOMER will appear several times in the SourceFile path, each time with another of its related INVOICE records. A report containing only the following detail line:

```
DETAIL 1 (CUST-NUMBER SHORT, INVOICE-NUMBER)
```

would print the following:

```
CUST          INVOICE
NUMBER        NUMBER
**********    *******
1620921286    SC20221
              SC20344
              SC20401
2153522440    SC41532
              SC43456
2248374765    SC10293
```

Example 2: VIA Index-name Path

```
SOURCEFILE DEMOPCB2
    PATH (CUSTOMER VIA CNAMINDX
    ,INVOICE VIA CUSTOMER)
```

The SourceFile will return the same information as the first example. As in the first example, the information obtained from the IMS segments will be grouped by customer, thanks to the Path construction. However, in this example, the customer records will be accessed in sequence by name. If the program reports required the data to be sorted in customer name order, accessing this IMS SourceFile through an Index, would eliminate the need for either a SourceFile or TargetFile sort.

Example 3: Associated record

```
SOURCEFILE DEMOPCB1
    PATH (INVOICE
    ,INV-REMARK VIA INVOICE OCCURS 5,
    ,ITEM VIA INVOICE)
```

In the path specification, the INV-REMARK records are associated with the INVOICE records by specifying the OCCURS option. In the event that there are fewer than five INV-REMARK records for a particular INVOICE, the system field INV-REMARK *SYS-PATH-COUNT* will indicate the number actually present. Refer to the *MetaMap Manager User Guide* for a description of the use of the SYS-PATH-COUNT system field.

The PATH command in this example will cause the system to construct a path that will allow for one INVOICE record, five INV-REMARK records, and one ITEM record. The first input path will contain the first INVOICE record stored in the database, the first five of its INV-REMARK records, and its first ITEM record. The next input path will contain the same INVOICE and INV-REMARK data, and the next ITEM record for the first INVOICE. This continues until there are no more ITEM records for the first INVOICE, at which point the next INVOICE record, its first five INV-REMARK records, and the first ITEM record for that INVOICE will be obtained.

The system will continue to construct paths in this way until the end of the INVOICE record chain in the database. Note that all of the SourceFiles shown in the examples thus far would have appeared in the MetaSuite Program listing with the Path Analysis Report, which is produced by the MetaSuite Generator. The Path Analysis Report produced for this example file would be as follows:

```
PATH   PATH RECORDS  ASSOCIATED RECORDS
****   ************  ********************
1      INVOICE       INV-REMARK(01 TO 05)
       ITEM
```

This Path Analysis Report shows that successive input paths will contain the hierarchy: INVOICE, ITEM. It also shows that up to five INV-REMARK records will be available for each INVOICE in the path. Note that whenever multiple occurrences of a record are specified, any references to the fields within those records must be have a subscript in order to identify the record occurrence desired.

For example, if it were desired to access a field named INV-REMARK-SEQ in the third occurrence of the associated INV-REMARK record, the reference would be coded:

```
INV-REMARK-SEQ (3) or INV-REMARK-SEQ (X)
```

where X is a numeric field having the value 3.

Example 4: Multiple Paths

```
SOURCEFILE DEMOPCB1
    PATH (INVOICE
    , INV-REMARK VIA INVOICE
    , ITEM VIA INVOICE)
```

In the SourceFile, the INV-REMARK record is implicitly related to the INVOICE record, forming the first path, and the ITEM record is explicitly related to INVOICE, forming the second path. The path analysis report for this file would be as follows:

```
PATH    PATH RECORDS        ASSOCIATED RECORDS
****    ************        ******************
1       INVOICE
        INV-REMARK
2       INVOICE
        ITEM
```

During execution, this path will always contain an INVOICE record. It may contain a new occurrence of either an INV-REMARK record or an ITEM record, but not a new occurrence of both. The system fields INV-REMARK SYS-PATH-COUNT and ITEM SYS-PATH-COUNT indicate which (if either) is present. Note that for a given occurrence of the INVOICE record, all of its INV-REMARK records will be returned before any of its ITEM records.

A program that prints INVOICE-NUMBER, INV-REMARK-SEQ, and ITEM-PROD-NUMBER values for this path only when those values change, would produce the following output:

```
           INV           ITEM
INVOICE    REMARK        PROD
NUMBER     SEQ           NUMBER
*******    ******        ********
SC20221    01
                         CCC11111
                         DDD22222
                         DDD22255
SC41533    01
           02
           03
                         CCC11144
                         DDD22222
```

Matching files

```
MATCH (match-key,…)
```

Match processing functions exactly as described for the SourceFiles in the *MetaMap Manager User Guide*. Note that the MATCH option can only be specified for Automatic SourceFiles, and should not be used with either the CONTROLLED or CONTROLLED BY options.

Controlled SourceFile

```
CONTROLLED
```

The *CONTROLLED* option indicates that access to the database is to be controlled using the MetaSuite GET command. Automatic, sequential access to the file will not be performed when the CONTROLLED option is specified.

When a CONTROLLED SourceFile is used, after each GET command you should check the *SourceFile SYS-IO-STATUS*, to detect whether the access has been successful. For information about the use of the SYS-IO-STATUS field, refer to the *MetaMap Manager User Guide*.

The success or failure of a GET command may also be determined by inspecting another system field named *SourceFile SYS-INTERNAL-STATUS*. In fact, the contents of this system field may be inspected at any time in a MetaSuite application. It contains the IMS status code returned from the last call to the database management system. It will be a 2-character field. The possible values for this system field are documented in IBM's *IMS/VS Application Programming: Designing and Coding manual*. The GET command examples below will alternate in the use of the SYS-IO-STATUS and the SYS-INTERNAL-STATUS fields.

A discussion of IMS considerations when using the GET command follows.

Controlled By SourceFile

```
CONTROLLED BY SourceFile-name KEY = control-key
```

The *CONTROLLED BY* option indicates that the records on one SourceFile will be accessed based on the information stored on, or derived from, another (controlling) SourceFile. *SourceFile-name* is the name of the (other) controlling SourceFile. *Control-key* is either a field on the controlling SourceFile or a GlobalField whose value is determined in the SourceFile input procedure for the controlling SourceFile.

When doing controlled retrieval using the CONTROLLED BY option, one SourceFile is read automatically (the control**ling** SourceFile), and the records are retrieved from another SourceFile (the control**led** SourceFile) according to the control-key. The composite record, referred to as a controlled set, is built consisting of records from both SourceFiles, and this composite record (controlled set) can then be used just as you would any single record.

The Initial SourceFile procedure for the CONTROLLED BY SourceFile cannot include commands that require a second pass: i.e., the Initial SourceFile procedure cannot request a SORT, EXTRACT, or PRE-PASS.

The SOURCEFILE command for the controlling SourceFile (*controlling-SourceFile-name*) must come before the SOURCEFILE command for the controlled SourceFile in the program. Also, the controlling SourceFile cannot itself be CONTROLLED, although it can be CONTROLLED BY another SourceFile. (You can nest CONTROLLED BY specifications up to a maximum of 20 SourceFiles).

The path of the controlling SourceFile must define a single path.

Note: If *control-key* is a field on the controlling SourceFile, the named field must be the lowest level of the SourceFile path hierarchy; that is, it must be the last non-occurs record defined for the path, or an OCCURS 1 record defined following the last non-occurs record.

Example 1: Controlled By

Assume that we have a file called CUSTOMER-CONTROL, which contains CUSTOMER-NUMBER-CONTROL values. We have another file, DEMOPCB1, (the DEMOPCB1 database file) that contains CUSTOMER, INVOICE and ITEM records. The CUSTOMER record STORAGE-KEY is the CUSTOMER-NUMBER. We want to write a program that prints the 1993 order information for customers specified in the control file.

Program Code

```
SOURCEFILE CUSTOMER-CONTROL
SOURCEFILE DEMOPCB1 CONTROLLED BY CUSTOMER-CONTROL
    KEY = CUSTOMER-NUMBER-CONTROL
    PATH (CUSTOMER, INVOICE VIA CUSTOMER,
ITEM VIA INVOICE)
REPORT 1
.
BEGIN RECORD INVOICE INPUT
IF INVOICE-PLACE-DATE LT 930101 EXCLUDE
BEGIN REPORT 1 INPUT
IF CUSTOMER SYS-PATH-COUNT EQ 0 -
    PUT (bad control-key value detail line) -
    EXIT
IF INVOICE SYS-PATH-COUNT EQ 0 -
    PUT (no invoice data detail line) -
    EXIT
IF ITEM SYS-PATH-COUNT EQ 0 -
    PUT (no item data detail line) -
    EXIT
PUT (full data detail-line)
```

Discussion

The CUSTOMER-CONTROL SourceFile will be read sequentially. The DEMOPCB1 SourceFile will be entered using direct access, and successive paths will be filled using sequential access.

The controlled sets available for report processing would be as follows:

```
CONTROL       CUSTOMER      INVOICE       ITEM
1620921286    1620921286    SC20221       CCC11111
                                          DDD22221
                                          DDD222255
                            SC20344       CCC11233

                            All INVOICE and ITEM data for
                            CUSTOMER 1620921286. If there
                            were more CUSTOMER records with
                            the same key value, they would
                            follow with all their INVOICE
                            and ITEM data.

2248374765    2248374765    SC41532       CCC22244
```

Note: The CUSTOMER-CONTROL records could be in any sequence; they do not need to be sorted. The Record Input procedure uses the EXCLUDE command to eliminate any non-1993 invoice data, and all other controlled sets are passed to report processing. The use of EXCLUDE speeds up processing, such that no ITEM records are accessed for an excluded INVOICE record.

The REPORT INPUT procedure checks for four conditions:

- missing CUSTOMER record (meaning that the CUSTOMER-CONTROL data did not correspond to an actual database record),
- missing INVOICE record (which occurs for a customer with no stored invoice information),
- missing ITEM record (which occurs for an invoice with no line-times -- probably an exception condition),
- complete information.

Example 2: Controlled by with Foreign Key

The second example shows the use of the CONTROLLED BY clause to process the "foreign key" stored in a data record. Let us assume for this example that the INVOICE record in the DEMOPCB1 database file is stored with a field called INVOICE-WRITTEN-BY, which holds a copy of the IMS storage key of a SALESPERSON record. INVOICE-WRITTEN-BY is the foreign key. We want to produce a report listing all customers and their invoices, along with the name of the salesperson who wrote each invoice. We want to process only those invoices written by salespeople in sales region 03.

Program Code

```
SOURCEFILE DEMOPCB1 PATH (CUSTOMER, INVOICE VIA CUSTOMER)
SOURCEFILE DEMOPCB1 PREFIX 'WRT-'
    CONTROLLED BY DEMOPCB1 KEY = ORDER-WRITTEN-BY
    PATH (WRT-SALESPERSON)
REPORT 1
.
BEGIN SOURCEFILE WRT-DEMOPCB1 INPUT
IF WRT-SALES-REGION NE 03 -
    EXCLUDE DEMOPCB1
IF INVOICE SYS-PATH-COUNT EQ 0 -
    PUT (no invoice data detail line) -
    EXIT
IF WRT-SALESPERSON SYS-PATH-COUNT EQ 0 -
    PUT (no salesperson data detail line) -
    EXIT
PUT (full data detail-line)
```

Discussion

The WRT-DEMOPCB1 SourceFile Input procedure checks the SALES-REGION value in the SALESPERSON record, and if it is not the desired region, it excludes the higher level SourceFile. This will cause the higher level SourceFile processing to retrieve the next INVOICE records, which in turn will cause the lower-level SourceFile processing to retrieve a new SALESPERSON record. Processing time will be improved because no controlled sets will be built for the unwanted sales regions. The REPORT INPUT procedure checks for three conditions:

1. missing INVOICE record (which occurs for a customer with no stored invoice information),
2. missing SALESPERSON record (which means that the ORDER-WRITTEN-BY information is incorrect -- probably an exception condition),
3. complete information.

Note: The procedural code looks exactly the same as it would if there were a link between INVOICE and SALESPERSON (and they were specified together in the PATH clause). The report output would also look exactly the same as if the records were related by a link.

5.4. Procedural Commands

Procedural commands tell MetaSuite what, if any, special processing is to be performed. Procedural code for a program that accesses an IMS database can include any of the procedural commands described in *MetaMap Manager User Guide Guide*. In certain cases, these commands differ in their use with IMS SourceFiles, as described below.

5.5. Command Summary

The MetaSuite commands used to access an IMS database are summarized below, then discussed individually:

This Command...	Is Used to...
EXCLUDE	Bypass processing of the current record and exit from the current procedure. You can exclude the current record (no subordinate records will be retrieved), the current path, or the current path from a controlling SourceFile.
GET	Read records from the database.
START	Position a SourceFile at a particular record before beginning access.

5.6. EXCLUDE

The following describes the EXCLUDE command and its use in accessing IMS databases.

Command Syntax

EXCLUDE [*record-name* | *SourceFile-name*]

Usage

If the SourceFile path contains multiple records, then the EXCLUDE command when used within a Record Input procedure allows you to bypass the accessing and processing of lower-level records in a path hierarchy, and to bypass the building of unwanted paths. As a slightly less efficient alternative, the EXCLUDE command can be used within a SourceFile Input procedure with the *record-name* option.

If the SourceFile is CONTROLLED BY, when an EXCLUDE command with the *SourceFile-name* option is executed, the generated program will skip all records in the controlled set from the named SourceFile down, and will build a new set of controlled records starting with the excluded SourceFile-name.

Use of the EXCLUDE command in any of the ways mentioned above will improve processing time because the overhead of accessing unwanted records and constructing unwanted sets of records is eliminated.

Basic Example

The discussions below refer to the following statements, and paths that could be built:

```
SOURCEFILE WIDGET-SALES-DB
PATH (CUSTOMER, BILL-TRANS VIA CUSTOMER)
REPORT 1
DETAIL 1 (CUSTOMER-NUMBER SHORT, BILL-NUMBER)
```

The following report (with the path number shown to the right) might be produced:

```
CUSTOMER       BILL
NUMBER         NUMBER
********       *******    (path number)
123 6 204      2948 33    (1)
               2993 27    (2)
               3842 46    (3)
206 3 412      3384 77    (4)
               3746 83    (5)
               4216 72    (6)
299 4 301      2981 92    (7)
               1293 09    (8)
303 3 009      3736 62    (9)
```

Nine paths were constructed from the thirteen records which were accessed (four CUSTOMERs and nine BILL-TRANSs).

Bypassing an individual path

To avoid reading unwanted subordinate records, and bypass building of unwanted paths, use the EXCLUDE command in a Record Input procedure. There can be one Record Input procedure for each record named in a path. When a record is accessed, the Record Input procedure for that record (if you've coded one) is processed before any additional records are accessed for that path. If the record is excluded, no records that are subordinate to the excluded record are accessed.

Syntax and use of the REPORT INPUT procedure is described in the *MetaMap Manager User Guide*. Briefly, a Record Input procedure is processed whenever the named record is accessed, before the completed path is processed by the SourceFile or TargetFile procedure. A Record Input procedure is coded immediately following any SourceFile procedures for the SourceFile to which the record is defined. A Record Input procedure begins as follows:

```
BEGIN RECORD record-name INPUT
```

Example

Given the basic example and paths above: to select (in the most efficient way) only customers with account numbers beginning with the digit 2, you might add the following code to the basic example:

```
BEGIN RECORD CUSTOMER INPUT
IF CUSTOMER-NUMBER NI (2000000 TO 2999999) EXCLUDE
```

The produced report would contain only the information shown in paths 4, 5, 6, 7, and 8. In the case of paths 1 and 9, the CUSTOMER record only would be accessed; since it is excluded in a Record Input procedure (for that record), no BILL-TRANS records are accessed for those customers. Paths 2 and 3 are never built. This is efficient because the least possible number of records have been accessed and the least possible number of paths have been built.

The same report results could have been obtained by using that same IF command in a TargetFile or SourceFile Input procedure. However, all records shown in the basic example would have been accessed and all paths built. This is dramatically less efficient than using a Record Input procedure for the same processing.

Bypassing Unwanted Paths and Subordinate Records

record-name

To bypass the building of unwanted paths, which are based on information in a completed path, use the EXCLUDE *record-name* command in a SourceFile Input procedure. The current path will be excluded, and no more paths will be built for the excluded record. The difference between this technique and excluding the record in a Record Input procedure is that in the latter, the path is not completed if the record is excluded (i.e., no subordinate records are read), while in the former the path is completed before the exclusion. In both cases, no further paths are built for the excluded record.

Example

Given the basic example and paths above: to select (in the most efficient way) only customers with account numbers beginning with the digit 2, you might add the following code to the basic example:

```
BEGIN SOURCEFILE WIDGET-SALES-DB INPUT
IF CUSTOMER-NUMBER NI (2000000 TO 2999999)-
EXCLUDE CUSTOMER
```

As in the previous example, the report produced would contain only the information shown in paths 4, 5, 6, 7, and 8, and paths 2 and 3 would not be built. Paths 1 and 9 would be built completely, with both CUSTOMER and BILL-TRANS records accessed, and then excluded. This is less efficient than using a Record Input procedure to bypass paths because in paths 1 and 9 the BILL-TRANS record is accessed unnecessarily.

Bypassing CONTROLLED BY Records

```
SourceFile-name
```

EXCLUDE *SourceFile-name* is applicable only in SourceFile Input procedures. The *SourceFile-name* option is used when one SourceFile is Controlled by another SourceFile, to identify the controlling SourceFile data in which you are no longer interested. *SourceFile-name* must be the name of a controlling SourceFile (defined through the CONTROLLED BY option for the SOURCEFILE command).

Example

Assume that a program contained the following statements:

```
SOURCEFILE DEMOPCB1
PATH (CUSTOMER, INVOICE VIA CUSTOMER)
SOURCEFILE DEMOPCB1 PREFIX 'WRT-'
CONTROLLED BY DEMOPCB1 KEY = ORDER-WRITTEN-BY
PATH (WRT-SALESPERSON)


REPORT 1
DETAIL 1 (CUSTOMER-NUMBER SHORT, INVOICE-NUMBER, -
    WRT-SALES-REGION, WRT-SALESPERSON)
```

The following report (with the path number shown to the right) might be produced:

```
CUST          INVOICE     WRT            WRT
NUMBER        NUMBER      SALESPERSON   (set-
***********   *******     ***********    number)
1620921286    SC20221     JONES         (1)
              SC20344     BLACK         (2)
              SC39374     REYES         (3)
2073849495    SC49483     CHANG         (4)
              SC25342     SMITH         (5)
```

```
             SC47365      KELLY            (6)
2994301234   SC36254      GABLE            (7)
             SC41092      HERON            (8)
3033009281   SC20982      HAMON            (9)
```

Nine controlled sets were constructed from the 22 IMS database records that were accessed (four CUSTOMERs, nine INVOICEs and nine WRT-SALESPERSONs). If you wanted to process salespersons from the southwest region only, you would add the following procedural commands to the program:

```
BEGIN SOURCEFILE WRT-DEMOPCB1 INPUT
IF WRT-SALES-REGION NE 03 -
    EXCLUDE DEMOPCB1
```

The WRT-DEMOPCB1 SourceFile Input procedure checks the SALES-REGION value in the SALESPERSON record, and if it is not the desired region, it excludes the higher level SourceFile. This will cause the higher level SourceFile processing to retrieve the next INVOICE record, which in turn will cause the lower-level SourceFile processing to retrieve a new SALESPERSON record.

This facility allows you to exclude unwanted controlled sets based on data in any portion of the controlled set. The report output would consist of the same as the above except that it would include only the sales persons from region 3.

5.7. GET

The following describes the GET command and its use in accessing IMS databases.

Command Syntax

```
GET {record-name | SourceFile-name}
    [KEY = keyfield-value]
```

Usage

The *GET* command is used to read records in an IMS database from within procedural code.

Except as described below, the options of the GET command, when used with an IMS database, are the same as for non-database SourceFiles.

Note that the success or failure of a GET command can be determined by inspecting the contents of the system field named *SourceFile* SYS-IO-STATUS. For information about the use of the *SYS-IO-STATUS* field, refer to the *MetaMap Manager User Guide*. The success or failure of a GET command may also be determined by inspecting another system field named *SourceFile* SYS-INTERNAL-STATUS.   In fact, the contents of this system field may be inspected at any time in a MetaSuite application. It contains the IMS status code returned from the last call to the database management system. It will be a 2-character field. The possible values for this field are documented in IBM's *IMS/VS Application Programming: Designing and Coding manual*. The GET command examples below will alternate in the use of the *SYS-IO-STATUS* and the *SYS-INTERNAL-STATUS* fields.

A discussion of IMS considerations when using the GET command follows.

Identifying the Record(s) to be Read

`{record-name | SourceFile-name}`

Required. You must specify either the *SourceFile-name* or the entry *record name* from the SourceFilePath.

Specifying the Access Key Value

KEY = *keyfield-value*

Optional. The *KEY* option specifies that MetaSuite is to retrieve the record(s) based on a keyfield value. *Keyfield-value* must be a literal or the name of a field of the same general data type (alphanumeric or numeric) as the access keyfield of the record to be obtained.

If the GET command names a record, the record must either have a storage-key or must be an indexed record. If the GET command names a SourceFile, the entry record in the SourceFilePath must have a storage-key or must be an indexed record. In the case of an indexed entry record, the VIA option must be included in the PATH specification, to name the index. *Keyfield-value* is the Storage-key or index-field-name value, as appropriate.

Combining SOURCEFILE and GET Command Syntax Options

The processing performed by MetaSuite for a GET command depends on the combination of GET command options specified for the IMS SourceFile being accessed, as summarized below.

GET KEY	VIA Index	GET retrieves...
NO	NO	The next occurrence of the entry record, as stored in the database. The path is refilled.
NO	YES	The next occurrence of the entry record in index set sequence. The path is refilled.
YES	NO	The (CALC) entry record occurrence containing the specified key value in its storage-keyfield. The path is refilled.
YES	YES	The entry record occurrence containing the specified key value in its index-name-field. The path is refilled.

Example 1: Retrieving Random of a record

```
SOURCEFILE DEMOPCB1 CONTROLLED
    PATH (CUSTOMER)
.
GET CUSTOMER KEY = '2903837698'
IF DEMOPCB1 SYS-IO-STATUS EQ SYS-ERROR -
    CUST-NAME = 'NOT FOUND'
```

In this example, the GET command will retrieve the CUSTOMER record using the specified storage-key value. Note that the CUSTOMER record was defined to the MetaSuite MetaStore with CUST-NUMBER, a 10-character alphanumeric field, specified as the *storage-keyfield*. Note also that, in the event that the CUSTOMER record with the desired storage-key is not found in the database, the name will print out as 'NOT FOUND'.

Example 2: Retrieving an Indexed Record

```
SOURCEFILE DEMOPCB2 PATH (CUSTOMER VIA CNAMINDX)
.
GET CUSTOMER KEY = 'HUDSON RIVER SPRINGS'
IF DEMOPCB2 SYS-INTERNAL-STATUS EQ 'GE' -
    CUST-NAME = 'NOT FOUND'
```

In this case, the CUSTOMER record will be accessed using the *index-keyfield*, CUST-NAME. Again, if the CUSTOMER record is not found, the name will print out as 'NOT FOUND'.

Example 3: Retrieving a Path of Records

```
SOURCEFILE DEMOPCB1 CONTROLLED
    PATH (CUSTOMER, INVOICE VIA CUSTOMEROCCURS 10)
.
GET CUSTOMER KEY = '2903837698'
IF DEMOPCB1 SYS-IO-STATUS EQ SYS-ERROR -
     CUST-NAME = 'NOT FOUND'
```

In this example, the same entry record will be obtained as was obtained in the first example, but in addition, up to 10 INVOICE records (beginning with the first) for the desired customer will be obtained. The number of INVOICE records actually obtained can be determined by examining the contents of the system field INVOICE SYS-PATH-COUNT.

5.8. START

The following describes the START command and its use in accessing IMS databases.

Command Syntax

```
START {record-name | SourceFile-name}
KEY = start-key
```

Usage

The *START* command is used to begin database access at a particular indexed record, by specifying a value for the index-field-name for the record. In this way, you can bypass the preceding records in the index.

Be aware that it is very easy to put your program into an infinite loop through improper use of the START command. See the discussion of the START command in the *MetaMap Manager User Guide* for more on this.

Identifying the Record or Subschema

```
{record-name | SourceFile-name}
```

Required. *SourceFile-name* or the entry *record name* from the SourceFilePath may be specified. The entry for the SourceFile being started must be the entry record for the PCB in use, must be indexed and include the VIA index-name option, to name the index you want to use.

Specifying the Starting Position

```
KEY = start-key
```

Required. The *KEY* option specifies the index key value less than or equal to the key value of the first record to be processed. *Start-key* may be either a literal or the name of a field of the same general data type (alphanumeric or numeric) as the index-field-name for the index to be used.

Example

Assume that it is desired to access all customers in our sample database whose company name begins with the letter "S". The following program could be coded:

```
SOURCEFILE DEMOPCB2
    PATH (CUSTOMER VIA CNAMINDX, INVOICE VIA CUSTOMER)
REPORT 1
DETAIL 1 (CUST-NAME SHORT, INVOICE-NUMBER)
BEGIN SOURCEFILE DEMOPCB2 INITIAL
START CUSTOMER KEY = 'S'
BEGIN SOURCEFILE DEMOPCB2 INPUT
IF CUST-NAME GE 'T' HALT SOURCEFILE
EXCLUDE
```

The report would access and list only those customers whose name begins with the letter "S". By coding the START command in the Initial SourceFile procedure, the database is "positioned" at the first CUSTOMER record meeting the START criteria. The IF command in the SourceFile Input procedure halts processing when the first CUSTOMER whose name does not begin with "S" is encountered.