IDMS File Access Guide

Release 8.1.3

November 2013

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

Table of Contents

Chapter 6 - Programming With MetaSuite File-Access (IDMS) 26

CHAPTER 1
About This Manual

MetaSuite file access for IDMS is intended for users with some experience with MetaSuite. It provides the information you need to use the MetaSuite IDMS Database File Access, including discussions on CA-IDMS concepts, defining a CA-IDMS 'database' for use with MetaSuite and using MetaSuite commands to access CA-IDMS databases.

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

1.1. Prerequisites

Readers are expected to be familiar with CA-IDMS.

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

MetaSuite File Access Overview

Because most MetaSuite commands are independent of the environment in which MetaSuite operates, only those commands that pertain to a CA-IDMS database definition and access are described in this supplement. Additional chapters in this manual include:

Topic	Description
CA-IDMS Concepts and Terminology	Presents a brief overview of the concepts and terminology you need to have to work with a CA-IDMS database.
About the MetaSuite IDMS Interface	Introduces the facilities of the MetaSuite IDMS Database Interface
Defining a CA-IDMS Database to MetaSuite	Tells you how to provide MetaSuite with the definitions of IDD. These are the definitions necessary to process data in a CA-IDMS database.
Programming with the IDMS File Access	Tells you how to use the MetaSuite commands that access information stored in a CA-IDMS database.
Reference to CA-IDMS DML Commands	Tells you how to embed CA-IDMS DML commands in a MetaSuite IDMS program.
Collect from IDD using MetaStore Manager	Tells how you can use the MetaStore Manager to produce MetaSuite file definitions for IDMS definitions out of IDD.
IDMS Samples	Some worked out MetaSuite MDL and MSL samples

CHAPTER 3

CA-IDMS Concepts and Terminology

3.1. Overview

IDMS is the database management system (DBMS) distributed by Computer Associates. In a CA-IDMS environment, data is stored in one centralised location, and is defined outside the scope of the application programs that use the data.

This chapter presents an overview of the CA-IDMS concepts and terms you should know before using MetaSuite to access a CA-IDMS database. It is broken down as follows:

- CA-IDMS data structures discusses the CA-IDMS database structures and objects that are pertinent to MetaSuite processing
- Accessing information discusses general database concepts and access considerations for application programs.

For more information, see CA-IDMS documentation from Computer Associates.

3.2. CA-IDMS data structures

This section describes the main CA-IDMS objects and related topics. The topics covered are:

- Records
- Location Modes
- Indexes
- Data structure diagrams
- Sets
- Set Linkage
- Set order
- Hierarchical data structure
- Network data structure

Each topic is described separately below.

3.3. Records

The basic unit of information that you retrieve from a database is a record occurrence. A record occurrence is a collection of related data items, or fields. A record type defines the format of similar record occurrences. There can be many record types in a database, each with its own definition.

Record types are connected logically by sets. A single record type can belong to many sets, or none. Each set defines a different logical relationship between record types. (More on sets later)

In MetaSuite file access for IDMS:

- A CA-IDMS record type is defined and referred to as a RECORD.

Location Modes

Each record is stored in a specific area of the database, using one of the following location modes to determine its physical location:

- DIRECT -- Records are stored according to a suggested database key, provided by the application program that stores the record.
- CALC -- Records are stored according to the hashed value or one or more key data items. This key value is called a CALC-key.
- VIA -- Member records for each set occurrence are stored physically near each other.

CA-IDMS assigns a unique Database key to each record when it is stored, and provides that key to the program for later use.

Indexes

An index provides ordered access to record occurrences, based on the value of a data item(s) in the record. A record type can have one or more indexes, or none.

For example, a customer record might be stored with the CALC location mode, where the CALC-key is the customer number. An index can be defined for the record, based on the customer name, to provide another means of access to the customer records.

In MetaSuite file access for IDMS:

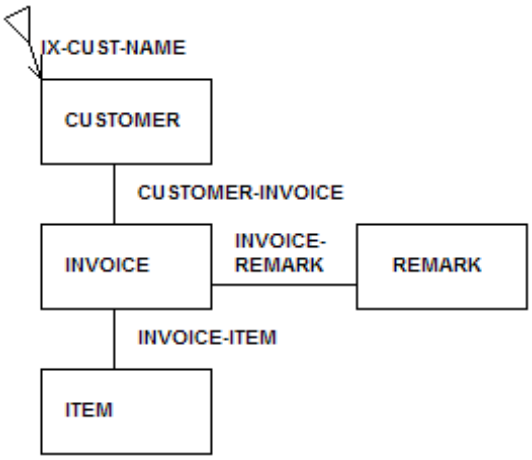
- A CA-IDMS index is defined and referred to as an INDEX.

Data Structure Diagram

A data structure diagram illustrates and documents the relationships among the records of a CA-IDMS database. These diagrams use the following conventions:

- A rectangle represents a database record type. Record-type rectangles are often subdivided to show specific information about the record.
- A circle represents a record occurrence.
- Lines connecting rectangles represent set types.
- Lines connecting circles represent actual relationships between record occurrences, within a set occurrence.
- A triangle represents an index, and is connected by a line to the rectangle that represents the indexed record type.

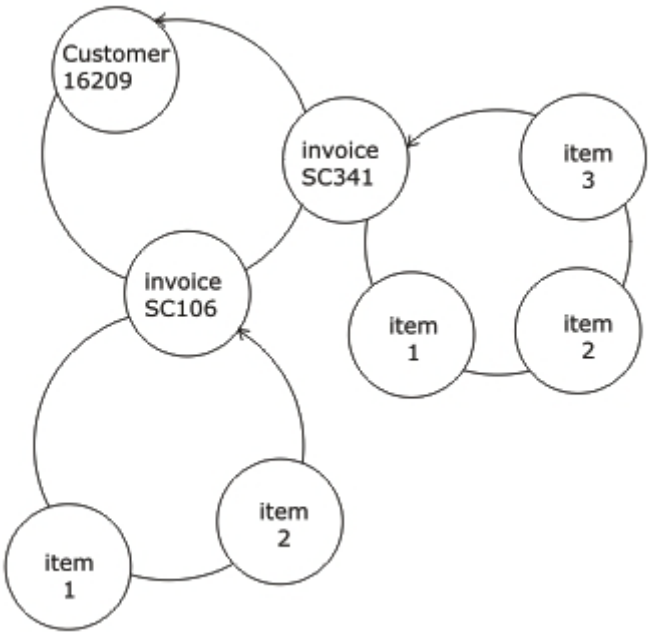
The following diagram illustrates a portion of a customer database. Customer, invoice, item, and invoice remark information is each stored separately, on records defined for that specific purpose.



The sets CUSTOMER-INVOICE, INVOICE-ITEM, and INVOICE-IREMARK relate, respectively, the CUSTOMER and INVOICE records, the INVOICE and ITEM records, and the INVOICE and IREMARK records. There is no direct relationship between CUSTOMER and ITEM records, or CUSTOMER and IREMARK records. Presumably, any application that uses these record combinations would also access the INVOICE record. Note that the INVOICE record belongs to three sets, while the other records belong to only one set each.

There is an index for the CUSTOMER record that indexes the records based on the customer names. (More on indexes later)

An occurrence of a CUSTOMER record, with its related INVOICE and ITEM records, might look like this:



Sets

Like records, sets have set types and set occurrences. A set type is the definition of a logical relationship between two or more record types, where one record type is the owner and the other type(s) is the member(s). The member record type(s) is logically subordinate to the owner record type. A set occurrence is a group of actual records that are associated by a logical set relationship.

For example, the INVOICE record is the owner of the INVOICE-ITEM set, in which ITEM records are members. Each INVOICE occurrence owns one or more ITEMs. Each ITEM represents a line item on the invoice that owns it.

For each set type, one set occurrence exists for each owner record occurrence. A set occurrence can have multiple member record occurrences. Each member record occurrence can be connected to a maximum of one set occurrence within a given set type. A set occurrence that has no member record occurrences is called an empty set.

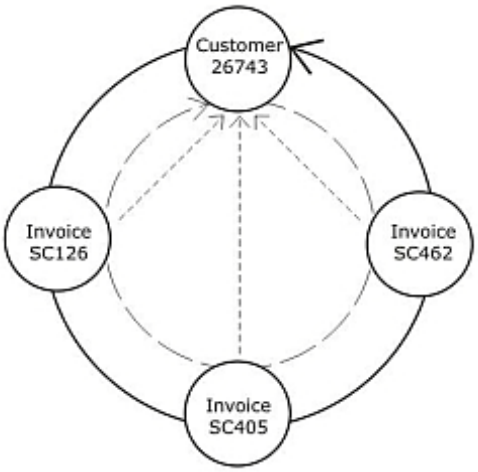
Note that a member record occurrence can exist disconnected to any set occurrence.

Set Linkage

The owner and members of a set occurrence are linked with one or more of the following types of pointers to link the record occurrences together within the set:

- NEXT (required for all sets). The owner points to the first member, the first member to the second member, and so on, with the last member pointing to the owner. This establishes a circular structure.
- PRIOR (optional). The owner points to the last member, the last member to the next-to-the-last member, and so on, with the first member pointing to the owner. This reverses the order established by NEXT pointers.
- OWNER (optional). Each member points to the owner.

The following diagram illustrates an occurrence of the CUSTOMER-INVOICE set, which is linked by all three types of pointers. The solid lines show the next pointers, the broken lines show the prior pointers, and the dotted lines show the owner pointers:



In MetaSuite file access for IDMS:

- A CA-IDMS SPF set (Sequential Processing Facility) is defined and referred to as an INDEX.
- A CA-IDMS non-SPF set is defined and referred to as a LINK

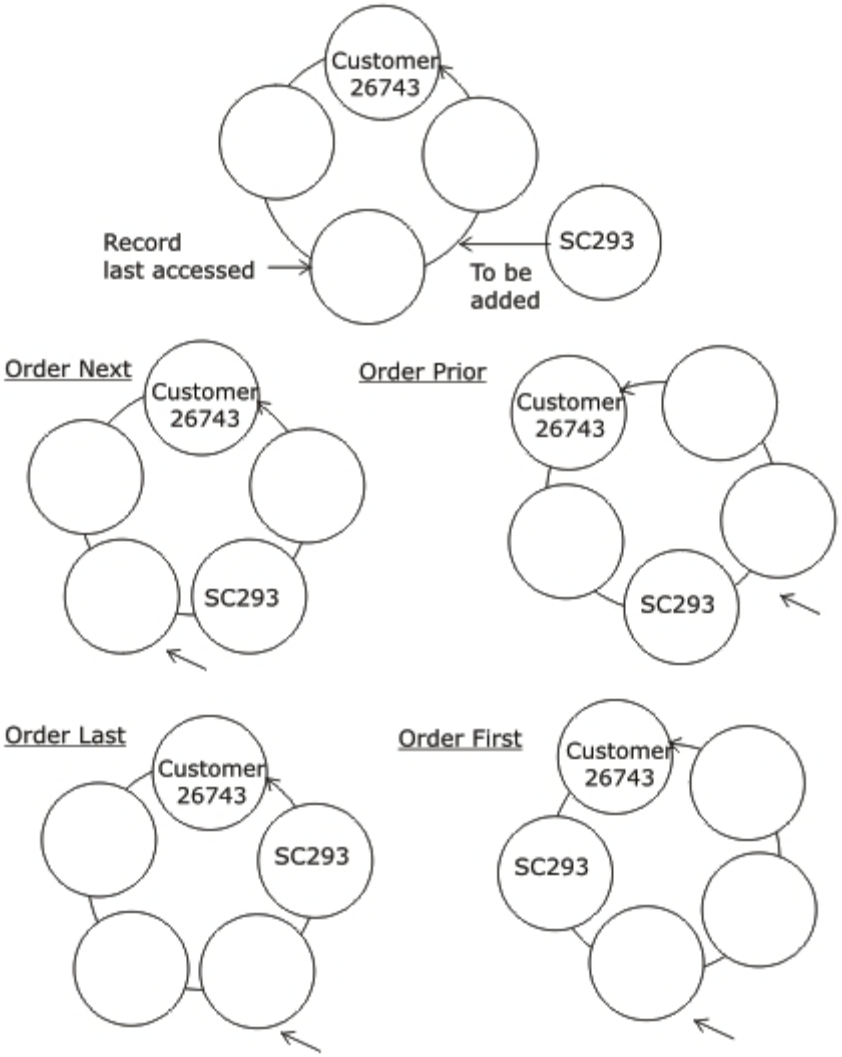
Set Order

The order defined for a set determines where a new member occurrence is linked into an existing set occurrence. Each set uses one of these ordering methods:

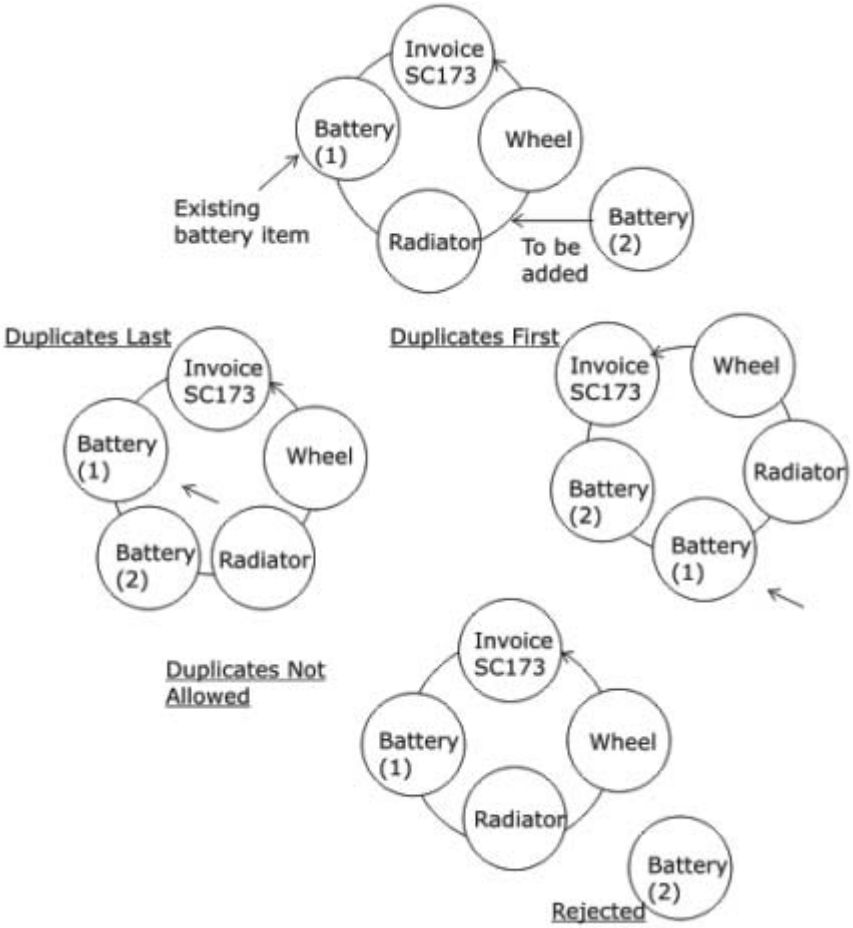
- FIRST - the new member is positioned immediately after the owner record, as the first member of the set.
- LAST - the new member is positioned immediately before the owner record, as the last member of the set.
- NEXT - the new member is positioned immediately after the current member of the set. (See "Accessing Information", later in this chapter, for more on currency.)

- PRIOR - the new member is positioned immediately before the current member of the set. (See "Accessing Information", later in this chapter, for more on currency.)
- SORTED - the new member is positioned according to the value of one of its fields (called a sort-key field), relative to the values of the same field in the other member records. Member records can be in ascending or descending order, with respect to the designated sort-key field. Records with duplicate sort-key values can be positioned first or last, or not allowed.

In the illustration below, invoice SC293 is added to the CUSTOMER-INVOICE set occurrence for customer 26743. The logical position of the new invoice within the set is shown for the next, prior, last and first set-order options. The program is positioned on the second member occurrence before the new invoice is inserted into the set (marked with an arrow).



To illustrate a sorted set, assume that the INVOICE-ITEM set is sorted in ascending order by the item name field. The following diagram shows how the duplicate item, BATTERY, is inserted for the duplicates options LAST, FIRST, and NOT ALLOWED:



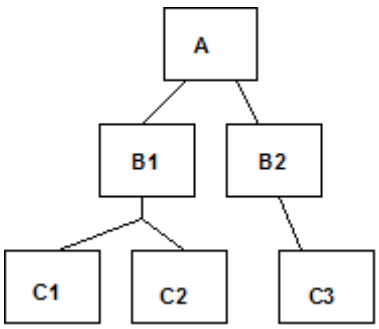
Data Structures

The relationship between records in a single set defines what can be viewed as a sequential structure: each record in the set is related to the records before and after it, creating a circular list.

A typical CA-IDMS database has many record types. Using sets to define the relationships among these record types, you can define either a hierarchical or a network data structure.

Hierarchical Structure

A hierarchy is a vertical structure where the member(s) of one set are owners of a second set, the members of that second set are owners of a third set, and so forth. Each record type might own multiple sets, thus creating the branching effect illustrated below:

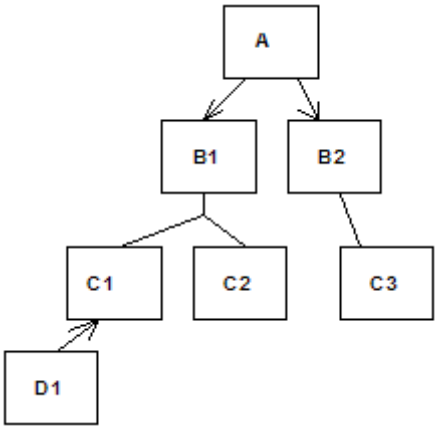


In the above diagram, the set that has the owner record type B1, and member record types C1 and C2, illustrates a multiple-member set. In this type of set, two or more record types are members of the set. The member record types are sufficiently related to be accessed together, but either are structured differently or must be separately accessed often enough to warrant separate record descriptions.

For example, you might create a multiple-member set in an employee database that relates an employee record to the company benefits provided to that employee. Each type of benefit (medical insurance, life insurance, pension, and so forth) would have its own record type. An occurrence of the set for an individual employee would include only those record types that are appropriate for that employee.

Network Structure

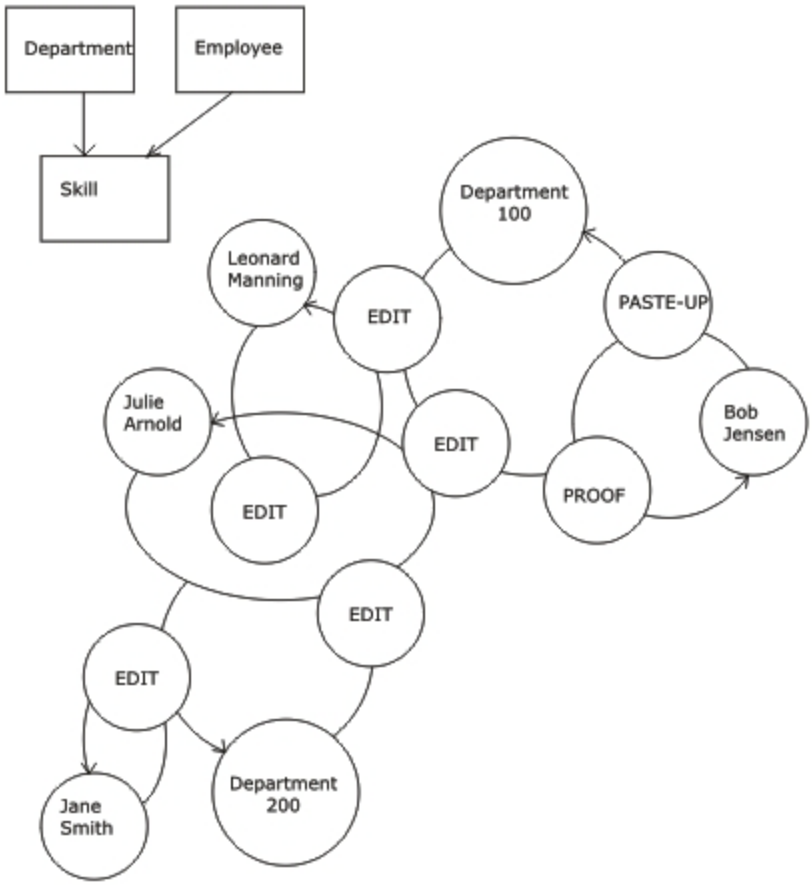
A network structure incorporates all the properties of a hierarchy and introduces one additional concept: records can participate as members in two or more sets. The following diagram illustrates a network structure:



A network structure provides the answer to the problem of shared membership: a single member record type that is owned by two (or more) other record types. This member record type, called a junction record, is associated logically with the owners in both sets, creating a many-to-many relationship (versus the one-to-many relationship in a hierarchy). Each junction record occurrence relates an owner record occurrence in one set with an owner record occurrence in the other set, and can be used to store data specific to that combination of owner records.

For example, assume that in an employee database there are DEPARTMENT records and EMPLOYEE records. Each department can have more than one employee, and each employee can work for more than one department. Assume that skills, required by the department and performed by an employee, provide the link between departments and employees.

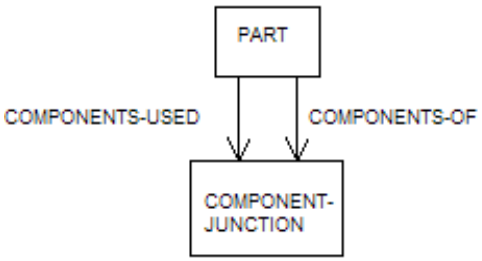
Department 100 requires EDIT, PROOF, and PAST-UP skills. Department 200 requires only EDIT skills. Jane Smith works for Department 200 (as an editor). Bob Jensen works for Department 100 (as a paste-up artist and proofreader). Julie Arnold and Leonard Manning work for both departments as editors. These relationships are illustrated on the next page:



There is a special case of a network structure, in which occurrences of the same record type are associated in a many-to-many relationship. This structure is called a bill-of-materials structure, because it represents the requirements of a manufacturing environment to associate manufactured goods with their component parts. Each component part might itself have other component parts, and so forth. (In fact, the manufactured good might be a component part of a still-larger item.)

This structure is effected by creating a member record that is related to the single owner through two set relationships: one pointing to subordinate components (components-used) and one pointing to larger items, of which the owner item is a component (components-of).

The created (junction) record might include information such as the quantity used. The following diagram illustrates a bill-of-materials structure:



3.4. Accessing Information

This section discusses some general concepts relating to CA-IDMS database access.

Integrated Data Dictionary (IDD)

A key component of a CA-IDMS database is the Integrated Data Dictionary (called the IDD). The IDD is itself a CA-IDMS database and contains the definitions of all the data and data relationships contained in the database. Definitions of non-database files can also be stored in the IDD.

All CA-IDMS users have at least one IDD. In a distributed or shared database environment, there may be multiple IDDs. In this environment, there is always a primary, or default, IDD that CA-IDMS uses if you do not name another IDD when you access the database.

During program processing, CA-IDMS uses the IDD for data definition information required to execute program requests.

Schema

The highest form of database definition information stored in the IDD is the schema definition. A schema completely describes a logical collection of records with their set relationships. There can be multiple schemas defined in the IDD, and multiple versions of the same schema definition.

A schema also includes information about the physical characteristics of the entities it defines. For records, this information includes the storage area within the database in which occurrences of the record are stored, and the location mode used to determine where to store each record within its area. For sets, this information includes the type of set linkage to be used and the order in which member records are to be stored in the set.

Subschema

An application view of a database is through a subschema. A subschema defines a subset of a schema, and may include all or some of the entities defined by the schema. A program can access only the records and fields that are defined to the subschema it uses to access the database.

Run-Unit

Each program that accesses a CA-IDMS database must name the subschema it wants to use when it first signs onto CA-IDMS. The sign-on process is called binding to CA-IDMS. Binding establishes the program as an individual run-unit within the CA-IDMS environment.

Once a run-unit is established, the program can begin to navigate the database to retrieve or store information.

Path

There are many ways to navigate a database structure. If a program needs to access more than one type of database record, a path through the records must be determined.

A path defines the way in which the program traverses the database: the choice of records to be processed and the relationships that join them. Each path begins with a specific record, or entry record, and proceeds through the database structure, from one record to another, using the set relationships. There are two types of paths: single path and multiple paths.

A single path has no branches. In the customer database, a single path might include the CUSTOMER, INVOICE, and ITEM records.

A multiple path has one or more branches. In the customer database, a multiple path might include the CUSTOMER, INVOICE, ITEM, and IREMARK records. the path branches at the INVOICE record, which owns both the ITEM and IREMARK records.

The choice of the entry record for a database path is often determined by the location mode of the records to be accessed. For example, a record that uses the CALC location mode often makes a good entry record, because it can be accessed directly, using its CALC-key value.

Currencies

To aid in successful navigation of the database, CA-IDMS maintains the Db-keys of the most recently accessed records, which are categorized as follows:

Currency Type	Description
Run-unit	The most recent record occurrence, of any type, accessed by the program is the current of run-unit.
Record type	The most recent record occurrence for each record type is the current of record type for that type.
Set	The most recent (owner or member) record occurrence in each set is the current of set for that set.
Area	The most recent record occurrence in each area is the current of area for that area.

CHAPTER 4

Using MetaSuite with a CA-IDMS database

This chapter introduces the MetaSuite facilities that allow you to access a CA-IDMS database from a MetaSuite application.

You should be familiar with the CA-IDMS concepts presented in the section CA-IDMS Concepts and Terminology (page 4) before reading this chapter.

To access a CA-IDMS database from a MetaSuite application, MetaSuite requires two things:

- Access to the CA-IDMS data definitions necessary to process the data in the CA-IDMS database. These definitions can come directly from IDD via the MetaStore Manager or prepared manually by creating an MDL (MetaSuite Definition Language) import – export text file.
- Specific MetaMap Manager commands to define the actual data retrieval requests for the CA-IDMS database. The MetaSuite Generator converts these MetaMap Manager commands into their CA-IDMS equivalents and produces a standard COBOL program with CA-IDMS statements.

The remainder of this chapter presents a brief overview of the MetaSuite facilities for providing access to the CA-IDMS data definitions and defining the data retrieval requests. This chapter also includes a description of the program generation process, as well as a summary of the differences between MetaSuite and CA-IDMS terminology.

4.1. MetaSuite and CA-IDMS 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 CA-IDMS terms are as follows:

MetaSuite	CA-IDMS
File	Schema
Record	Record Type
Field	Field
Index	CA-IDMS Index
Link	CA-IDMS set

The MetaSuite terms are described separately below, along with their CA-IDMS correspondences.

File

In MetaSuite file access for IDMS, a CA-IDMS Schema is defining a MetaSuite file. For each file, the default subschema within the schema to be used is specified on the file definitions through the DBNAME notion.

The file allows all data from the different records within the file to be treated as a logical record.

Record

In MetaSuite file access for IDMS, a CA-IDMS record type is known as a record.

One occurrence of a record type is referred to as a record occurrence. Note, though, that the term "record" can be used to mean either a record type or a record occurrence (that is, a row). The meaning in a given case depends on the context.

Field

In MetaSuite File Access for IDMS, a field on a record type is known as a field.

Index

In MetaSuite File Access for IDMS, a CA-IDMS index is known as an index. It will give you the possibility to determine yourself the access path towards the records.

Link

In MetaSuite File Access for IDMS, a CA-IDMS set is known as a link. You will use the links to determine how the relations have to be set between multiple records in a file.

Data Definition Facilities

MetaSuite provides two separate data definition facilities for use in the CA-IDMS environment:

- The MetaStore Manager collect option provides MetaSuite with access to the data definitions stored in IDD (exported through the use of a CA-IDMS punch) to define CA-IDMS objects to the MetaStore Manager (as records and fields).
- CA-IDMS dictionary files created manually in the MetaStore Manager.

Refer to the *MetaStore Manager Use Guide* for more information.

4.2. Programming Overview

MetaSuite programs have the same structure and report processing capabilities regardless of the file organization used. In other words, their structure and report processing capabilities are the same whether they are used to access DBMS or non-DBMS files.

Most of the MetaMap Manager commands with which you are already familiar can be used to process a CA-IDMS database.

When used with a CA-IDMS database, the MetaSuite Generator produces the native CA-IDMS calls for each MetaMap Manager SourceFile object that names a CA-IDMS file in a MetaSuite application. The generated native CA-IDMS calls will be dependent of the used path within the MetaMap Manager SourceFile. Multiple SourceFile objects for CA-IDMS files can be used in a single application program. The same matching, controlling, and buffering capabilities are used with a CA-IDMS "file" as with a non-database file.

The section Defining a CA-IDMS Database (page 17) presents detailed instructions on using the MetaMap Manager commands that access a CA-IDMS database.

CA-IDMS SourceFile path

In processing a program constructed with MetaSuite File Access for IDMS, the MetaSuite Generator converts the different components within the SourceFile path to the proper CA-IDMS native calls in the resulting COBOL source program.

The SourceFile object options for MetaSuite File Access for IDMS are described in the section Programming With MetaSuite File-Access (IDMS) (page 26).

Extended MetaSuite Facilities

The SourceFile object offers the following expanded file-based retrieval and processing, beyond the joining capabilities described above:

- File matching. The MATCH option of the SourceFile object allows CA-IDMS files (that is, data returned by relational) to be matched with CA-IDMS or non-CA-IDMS files.
- Controlled retrieval. The CONTROLLED option of the SourceFile object in conjunction with the GET command allows an individual row to be retrieved randomly.
- Controlled by retrieval. The CONTROLLED BY option of the SourceFile object allows sets of rows to be retrieved randomly.

What the Program Sees

What a MetaSuite application sees is a logical record (path) consisting of one row returned by the CA-IDMS call.

Multiple Databases

MetaSuite can access up to 100 files in one program. These files can include CA-IDMS, relational and other non-DBMS files.

CHAPTER 5

Defining a CA-IDMS Database

5.1. Overview

This chapter describes how to provide MetaSuite with access to the definitions that it needs to process data in a CA-IDMS database.

Note that the descriptions in this chapter use both the MetaSuite and CA-IDMS terminology for data entities, as appropriate. The correspondences between the MetaSuite and CA-IDMS terminology are discussed in detail in the section CA-IDMS Concepts and Terminology (page 4). These correspondences are summarized in the following table.

MetaSuite	CA-IDMS
File	A CA-IDMS schema
Record	A CA-IDMS record type
Field	A CA-IDMS field
Index	A CA-IDMS index
Link	A CA-IDMS set

Before you can access data in a CA-IDMS database, you must provide MetaSuite with access to the definitions necessary to process the data. CA-IDMS versions of the ADD FILE, ADD RECORD, ADD FIELD, ADD INDEX and ADD LINK commands are provided for defining CA-IDMS files to the MetaSuite MetaStore (see "Defining Databases Manually," later in this chapter).

However, an easier approach is to use the Collect File functionality of the MetaStore Manager to copy definitions out of a CA-IDMS schema IDD export, and then load the copied definitions into the MetaStore. We recommend the use of the collect functionality in the MetaStore Manager to define the CA-IDMS database to the MetaStore. For more information about this functionality, refer to Appendix A - MetaStore Manager Collect for CA-IDMS (page 60).

5.2. Defining Databases Manually

This section describes how to define a CA-IDMS database to MetaStore manually.

Before defining a CA-IDMS database to the MetaStore, you should obtain the necessary schema, record, field, link and set information from the CA-IDMS catalog.

Note: Use of the manual coding method is not recommended. It requires careful translation of the CA-IDMS definitions into MetaSuite definitions. As a result, it is more subject to error than using the *Collect File* option of MetaStore Manager.

Commands

The following table lists the commands used to manually code CA-IDMS data definitions.

MetaSuite Commands that Define CA-IDMS Data Structures

Command	Used to define
ADD FILE	A CA-IDMS schema
ADD RECORD	A CA-IDMS record type
ADD FIELD	A CA-IDMS field
ADD INDEX	A CA-IDMS index
ADD LINK	A CA-IDMS set

Each command is described separately below.

5.3. FILE

The *ADD FILE* command defines a CA-IDMS 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 CA-IDMS tables are described below.

The ADD FILE command will describe all records within a CA-IDMS schema.

Format

```
ADD FILE File-name TYPE IDMS
      SCHEMA Schema-name VERSION Schema-version
      DBNAME 'Subschema-name'
      [RULE Business-rule]
```

File-name

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

Schema-name

Required. *Schema-name* is the name of the CA-IDMS schema.

Schema-version

Required. *Schema-version* is the version of the CA-IDMS schema.

Subschema-name

Required. *Subschema-name* will specify the default subschema and optionally the default CA-IDMS database that is to be used within MetaMap Manager access. The name must be enclosed in single quotes. The format is 'Subschema-name[.Database-name]'.

Business-rule

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

Example

A CA-IDMS customer database is to be defined to the MetaStore. After examining the IDMSRPTS listings for this database, it is determined that the subschema, or "application view" for the database, that we want to use is named "CUSTSS01". An ADD FILE command would be coded as follows:

```
ADD FILE IDMSCUST TYPE IDMS
SCHEMA CUSTSCHM VERSION 1 DBNAME 'CUSTSS01'
```

5.4. RECORD

The *ADD RECORD* command defines a CA-IDMS record to the MetaStore. The general syntax for the ADD RECORD command is described in the *MetaMap Manager User Guide*. The options that refer to CA-IDMS records are described below.

Format

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

Usage

The ADD RECORD command defines a CA-IDMS database record to the MetaStore.

You can find the information you need to code on the ADD RECORD command in both the IDMSRPTS Subschema Data Dictionary Listing for the subschema, and the Subschema Record Description Listing for the record.

The Subschema Data Dictionary Listing provides information about a record in the following format:

```
RECORD: name ID: id VER: n TYPE: x LENGTH:  size
```

The Subschema Record Description Listing provides information about a record in the following format:

```
RECORD NAME      name
RECORD ID        id
RECORD VERSION   version
RECORD LENGTH    format
LOCATION MODE    mode
WITHIN           area-name
```

Record-name

Required. *Record-name* is the name of the record, as shown in the RECORD statement of the IDMSRPTS Subschema Data Dictionary Listing.

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 specified must be at least as large as the record LENGTH shown in the IDMSRPTS Subschema Data Dictionary Listing.

Storage-keyfield

Required for any record whose location mode is CALC. The LOCATION MODE line on the Subschema Record Description Listing appears as follows for a CALC record:

`LOCATION MODE CALC USING calc-key`

Storage-keyfield is the CALC-key name shown in the listing. If the location mode is VIA or DIRECT, the STORAGE-KEY option is not allowed.

Area-name

Required. *Area-name* is the WITHIN name on the IDMSRPTS Subschema Record Description Listing, for the record being defined.

Business-rule

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

Example

Assume that you want to add four record definitions to the MetaStore for the subschema CUSTSS01. The first step would be to examine the IDMSRPTS Subschema Data Dictionary and IDMSRPTS Subschema Record Description Listings for our sample database.

The record information on the Subschema Data Dictionary Listing might appear as follows:

```
RECORD: CUSTOMER   ID: 0611   VER: 002
      TYPE: I      LENGTH: 104
.
RECORD: INVOICE    ID: 0620   VER: 002
      TYPE: I      LENGTH: 40
.
RECORD: ITEM       ID: 0621   VER: 002
      TYPE: I      LENGTH: 226
.
RECORD: IREMARK    ID: 0622   VER: 002
      TYPE: I      LENGTH: 72
.
```

The record information on the Subschema Record Description Listing might appear as follows:

```
RECORD NAME           CUSTOMER
RECORD ID             0611
RECORD VERSION        002
RECORD LENGTH         FIXED
LOCATION MODE         CALC USING CUST-NUMBER
WITHIN                CUST-AREA
.
RECORD NAME           INVOICE
RECORD ID             0620
RECORD VERSION        002
RECORD LENGTH         FIXED
LOCATION MODE         CALC USING INVOICE-NUMBER
WITHIN                CUST-AREA
.
RECORD NAME           ITEM
RECORD ID             0621
RECORD VERSION        002
RECORD LENGTH         VARIABLE
LOCATION MODE         VIA SET INVOICE-ITEM
WITHIN                CUST-AREA
.
RECORD NAME           IREMARK
RECORD ID             0622
RECORD VERSION        002
RECORD LENGTH         FIXED
LOCATION MODE         VIA-SET INVOICE-ITEM
WITHIN                CUST-AREA
.
```

Using the information from these listings, the MetaSuite ADD RECORD commands would be coded as follows:

```
ADD RECORD CUSTOMER OF IDMSCUST SIZE 104 STORAGE-KEY CUST-NUMBER STORAGE-AREA CUST-
AREA
ADD RECORD INVOICE OF IDMSCUST SIZE 40 STORAGE-KEY INVOICE-NUMBER STORAGE-AREA
CUST-AREA
ADD RECORD ITEM OF IDMSCUST SIZE 226 STORAGE-AREA CUST-AREA
ADD RECORD IREMARK OF IDMSCUST SIZE 72 STORAGE-AREA CUST-AREA
```

The subschema reference CUSTSS01 will be done in the ADD FILE statement for IDMSCUST. The record sizes are taken from the LENGTH fields on the Subschema Data Dictionary Listing. The storage areas and storage keyfields are taken from the WITHIN and LOCATION MODE statements on the Subschema Record Description Listing.

5.5. INDEX

The *ADD INDEX* command defines a CA-IDMS index set to the MetaStore. The syntax for the ADD INDEX command is described below.

Format

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

Usage

A set is described on a CA-IDMSRPTS Subschema Set Description Listing, as follows:

```
SET          index-set-name
PRIVACY LOCK IS
OWNER        IXOWNER
MEMBER       member-record       ASC index-field-name
```

An OWNER record name of IXOWNER indicates that the set is to be defined as an index. If the owner record name is anything else, use the ADD LINK command to define the set.

Index-set-name

Required. *Index-set-name* is the name of a CA-IDMS set (identified by SET in the Subschema Set Description Listing).

Index-field-name

Required. *Index-field-name* is the name of the keyfield for the set, as shown following the MEMBER record name in the Subschema Set Description Listing.

Example

Assume that the IDMSRPTS Subschema Set Description Listing for the sample database shows the following information:

```
SET          IX-CUST-NAME
PRIVACY LOCK IS
OWNER        IXOWNER
MEMBER       CUSTOMER            ASC CUST-NAME
```

To define the set to the MetaStorSourceFile Path you would code:

```
ADD INDEX IX-CUST-NAME BASED ON CUST-NAME
```

Note that you must define the CUST-NAME field to the MetaStore before you can reference the IX-CUST-NAME index set in a MetaSuite application program.

5.6. LINK

The *ADD LINK* defines CA-IDMS sets to the MetaStore as link entities. The syntax for the ADD LINK command is described below.

Format

```
ADD LINK link-name
      FROM owner-record TO (member-record,...)
      [OPTIONAL]
```

Usage

A set is described in the IDMSRPTS Subschema Set Description Listing as follows:

```
SET          set-name
PRIVACY LOCK IS
OWNER        owner-record
MEMBER       member-record
```

Link-name

Required. *Link-name* is the set name (SET), as shown in the Subschema Set Description Listing for the subschema.

Owner-record

Required. *Owner-record* is the name of the OWNER record type, as shown in the Subschema Set Description Listing. If the owner record in the Subschema Set Description Listing is IXOWNER, the set must be defined to the MetaStore, using the ADD INDEX command.

Member-record

Required. Each *member-record* is the name of a MEMBER record type in the set being defined, as shown in the Subschema Set Description Listing. If the set has only one member record type, the parentheses may be omitted.

OPTIONAL

Optional. The OPTIONAL specification indicates that the participation of a given record type in the set is "optional"; that is, CA-IDMS allows the link between two record types to be either present or absent under user-defined conditions. If the MEMBER line of the IDMSRPTS Subschema Set Description Listing for the set contains the word OPTIONAL, you must include the OPTIONAL keyword here.

Example

To continue the definition of our sample CUSTSS01 database, assume that the IDMSRPTS Subschema Set Description Listing contains the following information:

```
SET          CUST-INVOICE
PRIVACY LOCK IS
OWNER        CUSTOMER
MEMBER       INVOICE
.
SET          INVOICE-ITEM
PRIVACY LOCK IS
OWNER        INVOICE
MEMBER       ITEM OPTIONAL
.
SET          INVOICE-IREMARK
PRIVACY LOCK IS
OWNER        INVOICE
MEMBER       IREMARK
.
SET          IX-CUST-NAME
PRIVACY LOCK IS
OWNER        IXOWNER
```

```
MEMBER       CUSTOMER            ASC CUST-NAME
```

Based on this information, the following ADD LINK commands would be coded for the CUSTSS01 database:

```
ADD LINK CUST-INVOICE FROM CUSTOMER TO INVOICE
ADD LINK INVOICE-ITEM FROM INVOICE TO ITEM OPTIONAL
ADD LINK INVOICE-IREMARK FROM INVOICE TO IREMARK
```

5.7. FIELD

The *ADD FIELD* command defines a record field as a field to the MetaStore. The syntax for the ADD FIELD command is described below.

Format

```
ADD FIELD field-name [OF { record | group-field  }]
      [POSITION start ]
      [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)]
      [RULE Business-rule]
```

Usage

The ADD FIELD command defines a CA-IDMS record field. The syntax is described in the *MetaMap Manager User Guide*, and is not repeated in this supplement.

CA-IDMS field-definition information appears in both the IDMSRPTS Subschema Data Dictionary Listing and the IDMSRPTS Subschema Record Description Listing.

Before generating MetaSuite application programs that reference the associated records, you must define any fields named as storage-keyfields on ADD RECORD commands or index-field-names on ADD INDEX commands, as well as any other fields referenced in your MetaSuite programs.

Example

In our example CUSTSS01 database, assume that the IDMSRPTS Subschema Data Directory Listing shows the following information for the CUSTOMER record:

```
NAME          LEVEL   STRT  LENGTH   TYPE   PICTURE
CUST-NUMBER   03        1      10     A/N     X(10)
CUST-NAME     03       11      20     A/N     X(20)
CUST-ADDRESS  03       31      40    GROUP
CUST-ADDR1    05       31      20     A/N     X(20)
```

```
CUST-ADDR2     05        51       20     GROUP
CUST-CITY      06        51       15      A/N      X(15)
CUST-ZIP       06        66        5      A/N      X(5)
.
```

The ADD FIELD commands for this record would be coded as follows:

```
ADD FIELD CUST-NUMBER OF CUSTOMER POSITION 1 SIZE 10 TYPE CHARACTER
ADD FIELD CUST-NAME POSITION 11 SIZE 20 TYPE CHARACTER
ADD FIELD CUST-ADDRESS POSITION 31 SIZE 40 TYPE CHARACTER
ADD FIELD CUST-ADDR1 OF CUST-ADDRESS POSITION 1 SIZE 20 TYPE CHARACTER
ADD FIELD CUST-ADDR2 OF CUST-ADDRESS POSITION 21 SIZE 20 TYPE CHARACTER
ADD FIELD CUST-CITY OF CUST-ADDR2 POSITION 1 SIZE 15 TYPE CHARACTER
ADD FIELD CUST-ZIP OF CUST-ADDR2 POSITION 16 SIZE 5 TYPE CHARACTER
    .
```

Note: The positions of subfields to the MetaStore are specified relative to the beginning of the group field, whereas in the IDMSRPTS Subschema Data Directory Listing, the positions of subfields are specified as absolute positions within the record.

CHAPTER 6

Programming With MetaSuite File-Access (IDMS)

6.1. Overview

This chapter describes how to use the MetaMap Manager commands that access information stored in a CA-IDMS database.

- Data source commands define the SourceFile, ExternalArray and GlobalField objects to be used during the program processing. For CA-IDMS SourceFiles, the SourceFilePath can specify how the CA-IDMS records need to be accessed within 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 Guide and in the MetaSuite Reference Guide.

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

Note that the descriptions in this chapter use the MetaSuite terminology exclusively. The correspondences between the MetaSuite and the CA-IDMS terminology are discussed in detail in the section Using MetaSuite with a CA-IDMS database (page 14). These correspondences are summarized in the following table.

MetaSuite	CA-IDMS
File	Schema
Record	Record type
Field	Field
Index	Index
Link	Set

6.2. Programming Considerations

When coding a program that accesses a CA-IDMS database, you should be aware of the considerations below.

Accessing the database

In general, you access a CA-IDMS database as you would access a non-database SourceFile.

You define each subschema that you want to access 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.

Besides the normal access methods, you can access a CA-IDMS database also through the known CA-IDMS DML commands. In this case the CA-IDMS database needs to be defined by a 'Manual' CA-IDMS SourceFile.

Note that the procedures you write for a SourceFile object can contain only one type of access command: either MetaSuite DML commands (to access Manual SourceFiles) or GET commands (to access Controlled SourceFiles), but not both.

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 CA-IDMS 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 MetaSuite Reference Guide for information about program processing sequence.

Navigating the Database

There are many ways to access a CA-IDMS database, some more efficient than others. The efficiency of your program processing can be determined by the entry record you choose to begin your database retrieval, and the methods you use to move from one record type to another within the path. 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 CA-IDMS database SourceFiles. Refer to the *MetaMap Manager User Guide* for the syntax of these commands.

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

```
EXCLUDE GET
EXIT HALT
SOURCEFILE START
```

In addition, there are commands specific to CA-IDMS that closely parallel the CA-IDMS Data Manipulation Language (DML) commands, both in their syntax and in their use. All of these commands are enclosed by the keywords EXEC-IDMS (translated to IDMS in the MSL, MetaSuite Specification Language) and END-EXEC, and are collectively referred to as MetaSuite DML commands.

Efficiency Considerations

When processing a CA-IDMS database, you can save both processing and I/O time by using the START and EXCLUDE commands.

The START command allows you to bypass unwanted entry records (and their related lower-level records).

If the SourceFilePath defined on the SourceFile accesses several SourceRecords, you can use the EXCLUDE command in a record input procedure, to bypass processing for lower-level records in the SourceFilePath hierarchy. This eliminates database accesses for the bypassed lower-level records.

If the SourceFile access is done through a 'CONTROLLED BY', you can use the EXCLUDE command to prevent the building of a controlled set or to skip processing for a controlled set already built. Processing time is improved by eliminating the overhead of constructing unwanted controlled sets.

Instructions to use the MetaSuite DML commands, the MetaSuite commands whose processing differs when accessing a CA-IDMS database, and commands that apply only to CA-IDMS databases appear in the remaining sections of this chapter.

6.3. SourceFile

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

Automatic SourceFile object:

```
SOURCEFILE SourceFile-name [PREFIX 'prefix']
SCHEMA schema-name VERSION version-number
DBNAME subschema-name
PATH (entry-record [VIA index-name]
[{,subordinate-record VIA link-name
      [OCCURS number times]} …])
[MATCH (match-key,…) ]
```

Manual SourceFile object:

```
SOURCEFILE SourceFile-name [PREFIX 'prefix']
SCHEMA schema-name VERSION version-number
DBNAME subschema-name
      { MATCH (match-key,…) | CONTROLLED }
MANUAL
```

Controlled SourceFile object:

```
SOURCEFILE SourceFile-name [PREFIX 'prefix']
SCHEMA schema-name VERSION version-number
DBNAME subschema-name
CONTROLLED
PATH (entry-record [VIA index-name]
[{,subordinate-record VIA link-name
      [OCCURS number times]} …])
```

Controlled By SourceFile object:

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

Each option is described separately on the following pages.

SourceFile-name

Names a SourceFile that has been defined to the MetaStore.

Prefix

Prefix-value 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.

Schema Name

Schema-name is the CA-IDMS schema that you want to use.

Version Number

Version-number is the version of the schema you want to use.

Subschema Name

Subschema-name is the GlobalField, whose value specifies the subschema and optionally the CA-IDMS database you wish to access. The value of the GlobalField is previously set to the subschema and database that is defined for the dictionary file in the MetaStore. Its value may be overwritten in a run-time parameter, but should not be altered in procedural code.

The *subschema-name* is previously defined as:

```
FIELD WK-SourceFile-name TYPE CHARACTER SIZE 17
INITIAL 'dbname'
```

With dbname as Subschema-name[.IDMS-Database-name]

PATH

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

The PATH option is part of the SOURCEFILE command.

It is mandatory for all IDMS SourceFiles except when the SourceFile is Manual. The SourceFilePath requests that the generated program constructs a single unit of data, called a "path", from multiple related records in the database. When this option is in effect, each time data is presented to the SourceFile initial procedure, SourceFile input procedure, or any report (or TargetFile) input procedure, it is the path of data from the entry record and its subordinate records that is presented rather than a single record.

When used with a subschema, the SourceFilePath identifies the particular database record types of interest and the navigational path(s) to be used to retrieve the records, and expresses the relationships between the records in hierarchical terms. Entry-record is the name of the "highest", or "first-level", record in the hierarchy; that is, the record that indicates the entry point for the path. Each subordinate-record is the name of a record related to the entry record or to another previously specified subordinate record.

Note that a link relationship must exist between each record and the next "lower" record in the path hierarchy.

The records named in the PATH specification are said to be either path records or associated records.

The hierarchical route through the records of the database, beginning with the entry record and ending with the lowest-level subordinate record, is called a path. For example, a path through our example database might begin at the CUSTOMER record, proceed to the INVOICE record, and end at the ITEM record. Each of these three records is a path record.

An associated record relates to a path record, but does not itself participate in the path. For example, you could associate the IREMARK record with the INVOICE record in the path example above. You define an associated record using the OCCURS option (more on this below, under "Identifying Associated Records").

The rest of this discussion is broken down to provide a detailed description of each PATH specification option, some examples of advanced path techniques, and a discussion of the Path Analysis Report for database SOURCEFILE statements.

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, MetaSuite retrieves the entry records in the sequence they were stored. For the CUSTSS01 subschema, the following PATH specification requests processing for all CUSTOMER records:

```
FIELD WK-IDMSCUST TYPE CHARACTER SIZE 17 INITIAL 'CUSTSS01'
SOURCEFILE IDMSCUST SCHEMA CUSTSCHM VERSION 1
      DBNAME WK-IDMSCUST
      PATH (CUSTOMER,...)
```

With the VIA option, MetaSuite retrieves the entry records in sequence by the index-name. The named index must be defined for the entry record, using the ADD INDEX dictionary command.

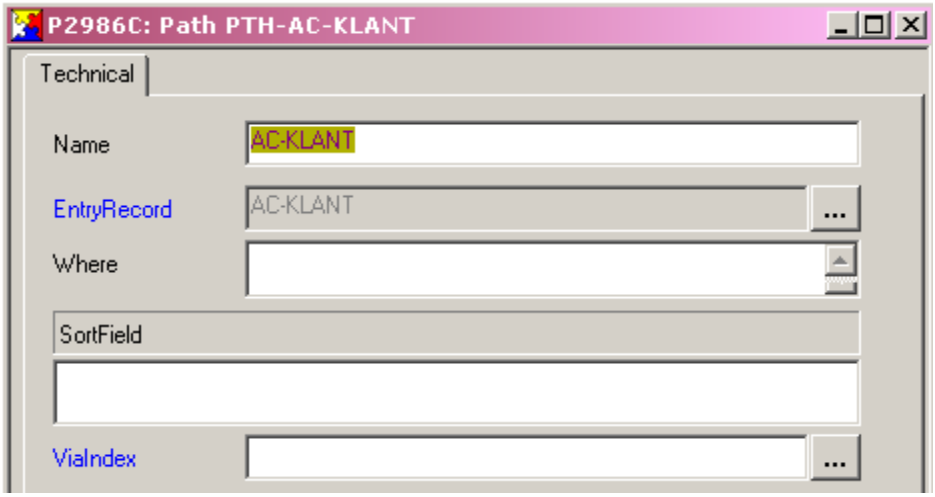
For example, to request that CUSTOMER records will be processed using the IX-CUST-NAME index, you would code:

```
FIELD WK-IDMSCUST TYPE CHARACTER SIZE 17 INITIAL 'CUSTSS01'
SOURCEFILE IDMSCUST SCHEMA CUSTSCHM VERSION 1
      DBNAME WK-IDMSCUST
      PATH (CUSTOMER VIA IX-CUST-NAME,...)
```

The CUSTOMER records are accessed in sequence by name. If the program reports require the data to be sorted in customer name order, use of this SOURCEFILE command would eliminate the need for either a SourceFile sort or a report/TargetFile sort.

MetaMap correspondence

The ViaIndex field is available in the PATH window.



Identifying Subordinate Records

subordinate-record VIA *link-name*

Optional. *Subordinate-record* names a record in the database that has a link relationship with the entry record or a previously specified subordinate record. There can be up to 15 subordinate records specified following the entry record.

Link-name names the relationship (set) that needs to be used to find the subordinate-record for an entry record.

Note that "subordinate" does not mean the record must be a member of a set owned by the previous record. The subordinate record may be either the owner or member record of a set containing the previously named record in the path.

For example, the following path specification requests CUSTOMER and INVOICE records:

```
FIELD WK-IDMSCUST TYPE CHARACTER SIZE 17 INITIAL 'CUSTSS01'
SOURCEFILE IDMSCUST SCHEMA CUSTSCHM VERSION 1
      DBNAME WK-IDMSCUST
      PATH (CUSTOMER, INVOICE VIA CUST-INVOICE)
```

The system retrieves a CUSTOMER record, then each of its INVOICE records, before retrieving the next CUSTOMER record. With this SOURCEFILE command, a report containing the following detail line:

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

might print the following:

```
CUST            INVOICE
NUMBER          NUMBER
************    *******
16209121286     SC20221
                SC20344
                SC20401
2153522440      SC41532
                SC43456
2248374765      SC10293
     .
```

Alternatively, you could code:

```
FIELD WK-IDMSCUST TYPE CHARACTER SIZE 17 INITIAL 'CUSTSS01'
SOURCEFILE IDMSCUST SCHEMA CUSTSCHM VERSION 1
      DBNAME WK-IDMSCUST
      PATH (INVOICE, CUSTOMER VIA CUST-INVOICE)
```

This SOURCEFILE command returns exactly the same information as the previous SOURCEFILE command, except that the input data is in the storage order of the INVOICE records rather than the CUSTOMER records. Depending on other program functions (such as sorting and record selection), relative record population sizes and densities in the database, and the internal configuration of the database, one or the other of these two PATH specifications might be more efficient. If efficiency is a consideration, consult with your systems staff for advice.

Let's add a third record to the PATH specification:

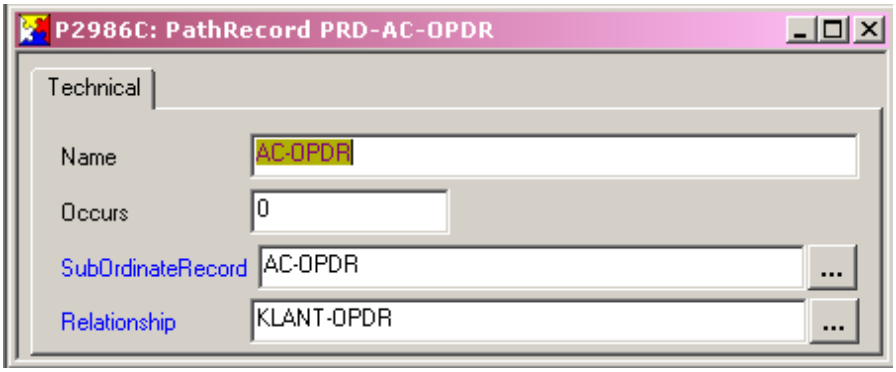
```
FIELD WK-IDMSCUST TYPE CHARACTER SIZE 17 INITIAL 'CUSTSS01'
SOURCEFILE IDMSCUST SCHEMA CUSTSCHM VERSION 1
      DBNAME WK-IDMSCUST
      PATH (CUSTOMER, INVOICE VIA CUST-INVOICE,
            ITEM VIA INVOICE-ITEM)
```

MetaSuite 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. These three records would be available to the MetaSuite program procedures as the first path of data.

MetaSuite would next attempt to obtain another ITEM record for the same INVOICE, and would return the new ITEM record, along with the old INVOICE and CUSTOMER. When there are no more ITEM records for the current INVOICE, MetaSuite would 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, MetaSuite would obtain the next CUSTOMER record and process its INVOICE and ITEM records as described above.

MetaMap correspondence

The record set definition (or LINK) must be specified in the Relationship field of the subordinate path.



Identifying Associated Records

`OCCURS` *`number-times`*

Optional. The OCCURS option identifies the subordinate record as an associated record. A link relationship must exist between the associated record and a preceding path record in the PATH specification. *Number-times* is a number from 1 to 32,767 that indicates the number of occurrences of the record you want to retrieve.

To determine the number of occurrences of the record actually retrieved after each database access, reference the system field record-name SYS-PATH-COUNT. See the MetaSuite Reference Guide for a description of the use of the SYS-PATH-COUNT system field.

Using the same CUSTOMER, INVOICE, ITEM path described above, assume that you would also like to access information from the first five IREMARK records for each INVOICE. You would code:

```
FIELD WK-IDMSCUST TYPE CHARACTER SIZE 17 INITIAL 'CUSTSS01'
SOURCEFILE IDMSCUST SCHEMA CUSTSCHM VERSION 1
      DBNAME WK-IDMSCUST
      PATH (CUSTOMER,INVOICE VIA CUST-INVOICE,
            IREMARK VIA INVOICE-REMARK OCCURS 5,
            ITEM VIA INVOICE-ITEM)
```

MetaSuite processes the CUSTOMER, INVOICE, ITEM path as described above, except that each time a new INVOICE record is obtained, up to five IREMARK records associated with the INVOICE record are also obtained. Specifically, the path contains one CUSTOMER record, one INVOICE record, the first five IREMARK records for the INVOICE, and one ITEM record.

MetaSuite automatically assumes that ITEM records are related to INVOICE records, because INVOICE is the last non-OCCURS (that is, path) record, which precedes ITEM.

Note that references to the fields in an OCCURS record must have a subscript, to identify the particular record occurrence desired. For example, if you want to access a field named IREMARK-SEQ in the third occurrence of the IREMARK record, you might code:

`IREMARK-SEQ (3) or IREMARK-SEQ (`*`field-name`*`)`

where field-name is a numeric field having the value 3.

Example 1: Multiple Path

Assume that you want to retrieve information from INVOICE, IREMARK, and ITEM records. IREMARK and ITEM records both have a link relationship with INVOICE. You might code:

```
FIELD WK-IDMSCUST TYPE CHARACTER SIZE 17 INITIAL 'CUSTSS01'
SOURCEFILE IDMSCUST SCHEMA CUSTSCHM VERSION 1
```

```
      DBNAME WK-IDMSCUST
      PATH (INVOICE,IREMARK VIA INVOICE-IREMARK,
            ITEM VIA INVOICE-ITEM)
```

This PATH specification defines a multiple path. The first path includes the INVOICE and IREMARK records. The second path includes the INVOICE and ITEM records.

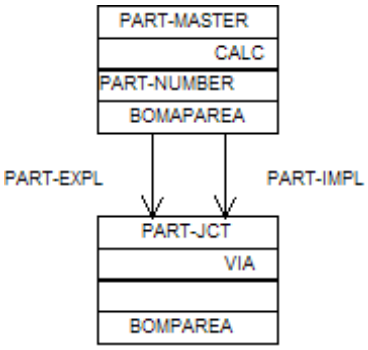
During execution, this path contains an INVOICE record, and either an IREMARK record or an ITEM record. You would use the system fields IREMARK SYS-PATH-COUNT and ITEM SYS-PATH-COUNT to determine which type of record is present.

Note that for a given occurrence of the INVOICE record, all of its IREMARK records are returned before any of its ITEM records. A program that prints INVOICE-NUMBER, IREMARK-SEQ, and ITEM-PROD-NUMBER values using this PATH specification might produce the following output:

```
             INV          ITEM
INVOICE      REMARK       PROD
NUMBER       SEQ          NUMBER
*******      ******       ********
SC20221      01           CCC11111
                          DDD22222
                          DDD22255
SC42533      01
```

Example 2: Bill-of-Materials Paths

For this example, assume that you have a bill-of-materials structure. For a CA-IDMS database, this structure is effected using two record types and two link types, as diagrammed here:



The MetaStore commands that define this structure are as follows:

```
ADD FILE IDMSBOMP TYPE IDMS SCHEMA BOMPSCHM VERSION 1
      DBNAME 'CUSTSS01'
ADD RECORD PART-MASTER OF (IDMSBOMP) SIZE 224
      STORAGE-AREA BOMPAREA STORAGE-KEY PART-NUMBER
ADD RECORD PART-JCT OF (IDMSBOMP) SIZE 16
      STORAGE-AREA BOMPAREA
ADD LINK PART-EXPL FROM PART-MASTER TO PART-JCT
      OPTIONAL
ADD LINK PART-IMPL FROM PART-MASTER TO PART-JCT
      OPTIONAL
```

The following MetaSuite SOURCEFILE command would be used to expand, or "explode", all part descriptions down to three levels in the structure:

```
FIELD WK-IDMSBOMP TYPE CHARACTER SIZE 17 INITIAL 'CUSTSS01'
SOURCEFILE IDMSBOMP SCHEMA BOMPSCHM VERSION 1
      DBNAME WK-IDMSBOMP
      PATH (PART-MASTER,
            PART-JCT VIA PART-EXPL,
```

```
            PART-MASTER VIA PART-IMPL,
            PART-JCT VIA PART-EXPL,
            PART-MASTER VIA PART-IMPL)
```

The path analysis report (discussed below) for this SourceFile would be as follows:

```
PATH     PATH RECORD     ASSOCIATED RECORDS
****     ***********     ********************
1        PART-MASTER     (01)
         PART-JCT        (01)
         PART-MASTER     (02)
         PART-JCT        (02)
         PART-MASTER     (03)
```

A program containing this SOURCEFILE command might use the following code to produce a three-level bill-of-materials explosion:

```
FIELD COMPONENT SIZE 12 TYPE CHARACTER
FIELD SUB-COMPONENT SIZE 12 TYPE CHARACTER
REPORT 1 PAGE (55,80)
DETAIL 1 (PART-NAME (1) SHORT, -
            COMPONENT SHORT, SUB-COMPONENT)
BEGIN REPORT 1 INPUT
CASE PART-MASTER SYS-PATH-COUNT -
      EQ 3 COMPONENT = PART-NAME (2) -
            SUB-COMPONENT = PART-NAME (3) -
      EQ 2 COMPONENT = PART-NAME (2) -
            SUB-COMPONENT = ' ' -
      ELSE (COMPONENT, SUB-COMPONENT) = ' '
```

Note that the SYS-PATH-COUNT field for the PART-MASTER record is checked to prevent non-current values of the PART-NAME field being printed in the report.

This program might produce the following output:

```
PART                COMPONENT         SUB
NAME                                  COMPONENT
***************     ***************   ******************
CHICKEN NOODLE SOUP CHICKEN SOUP BASE DEHYD CHICKEN BROTH
                                      WATER
                                      CUBED CHICKEN
                    NOODLE            BLEACHED FLOUR
                                      EGG
                                      VEG SHORTENING
                                      WATER
                    PACKAGE 112       CAN 322
                                      LABEL 123
                                      GLUE 224
CHICKEN RICE SOUP   CHICKEN SOUP BASE DEHYD CHICKEN BROTH
                                      WATER
                                      CUBED CHICKEN
                                      WHITE RICE
                    PACKAGE 114       CAN 322
                                      LABEL 124
                                      GLUE 224
```

The Path Analysis Report

Any time the PATH specification is coded, the program generator produces a Path Analysis Report, summarizing the paths that have been specified.

For the following SOURCEFILE command, which defines a single path:

```
FIELD WK-IDMSCUST TYPE CHARACTER SIZE 17 INITIAL 'CUSTSS01'
SOURCEFILE IDMSCUST SCHEMA CUSTSCHM VERSION 1
      DBNAME WK-IDMSCUST
      PATH (CUSTOMER,INVOICE VIA CUST-INVOICE,
            IREMARK OCCURS 5 VIA INVOICE-IREMARK,
            ITEM VIA INVOICE-ITEM)
```

the path analysis report would look like this:

```
PATH      PATH RECORDS     ASSOCIATED RECORDS
****      *************    **********************
1         CUSTOMER
          INVOICE          IREMARK (01 TO 05)
          ITEM
```

For the following SOURCEFILE command, which defines a multiple path:

```
FIELD WK-IDMSCUST TYPE CHARACTER SIZE 17 INITIAL 'CUSTSS01'
SOURCEFILE IDMSCUST SCHEMA CUSTSCHM VERSION 1
      DBNAME WK-IDMSCUST
      PATH (INVOICE,IREMARK VIA INVOICE-REMARK,
            ITEM VIA INVOICE-ITEM)
```

the path analysis report would look like this:

```
PATH       PATH RECORDS    ASSOCIATED RECORDS
****       *************   **********************
1          INVOICE
           IREMARK
2          INVOICE
           ITEM
```

Matching files

`MATCH (`*`match-key`*`,…) [MANUAL]`

The *match-key* allows you to view records from different SourceFiles simultaneously: match-key identifies the SourceField or GlobalField whose value is used for match processing.

Match processing functions exactly as described for the SourceFile command in the MetaSuite Reference Guide, with the exceptions noted below.

With the MANUAL option (on a Manual SourceFile), you must use MetaSuite DML commands to access the CA-IDMS database in the SourceFile initial procedure. These DML commands must be done on a SourceFile initial extract procedure, or on a SourceFile initial procedure when SortFields are defined on the SourceFile.

For example, to match a (non-CA-IDMS) SourceFile containing sales statistics with salesperson information in a CA-IDMS database, you might code:

```
FIELD WK-IDMSCUST TYPE CHARACTER SIZE 17 INITIAL 'CUSTSS01'
SOURCEFILE YTD-SALES MATCH (YTD-SALESPERSON)
SOURCEFILE IDMSCUST SCHEMA CUSTSCHM VERSION 1
      DBNAME WK-IDMSCUST
      PATH (SALESPERSON) MATCH (SALES-NAME)
```

Controlled SourceFile

```
CONTROLLED [MANUAL]
```

The CONTROLLED option indicates that all access to the SourceFile is through procedural code, using the MetaSuite GET command. The GET command allows you to retrieve one specified record or one path from the CONTROLLED SourceFile.

For example, to define a SourceFile that you want to access using the GET command, you might code:

```
FIELD WK-IDMSCUST TYPE CHARACTER SIZE 17 INITIAL 'CUSTSS01'
SOURCEFILE IDMSCUST SCHEMA CUSTSCHM VERSION 1
      DBNAME WK-IDMSCUST CONTROLLED
      PATH (SALESPERSON)
```

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. Please refer to the section Procedural Commands (page 39) for returned values of the SYS-IO-STATUS.

The CONTROLLED MANUAL option indicates that all access to the database using the SourceFile is through procedural code, using the MetaSuite DML commands. All the DML commands must be done from SourceFile procedure of another SourceFile, or from a TargetFile procedure. You can not add SourceFile procedures to CONTROLLED MANUAL SourceFiles. Please refer to the section MetaSuite CA-IDMS DML Commands (page 49) for more information on the DML commands.

Controlled By SourceFile

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

The CONTROLLED BY option indicates that the records in the database are to be accessed based on values in another 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.

You must define a SourceFile path to use the CONTROLLED BY option. The entry-record named in the SourceFilePath must be a CALC record (that is, defined with a storage-keyfield) or an indexed record (whose index-field-name was defined using the ADD INDEX directory command).

To process a CONTROLLED BY SourceFile, MetaSuite reads the controlling SourceFile and retrieves its records, using the control-key value. MetaSuite builds a composite record, called a controlled set, consisting of records from both SourceFiles.

The controlling SourceFile must define a single path. The path for either the controlled or controlling SourceFile can include associated records; that is, subordinate-records in the SourceFile path that include the OCCURS option.

The controlling SourceFile cannot be CONTROLLED itself, although it can be CONTROLLED BY another SourceFile. You can nest CONTROLLED BY specifications for use up to 20 SourceFiles.

If the control-key is a field on the controlling SourceFile, the named field must be in a record in the lowest level of the SourceFile path hierarchy. Specifically, the record must be the last non-occurring record, or an OCCURS 1 record that follows the last non-occurring record, as defined in the SourceFilePath.

Note that a CONTROLLED BY SourceFile can not have an Initial Prepass or an Initial Extract procedure, nor can it have SortFields defined. (that is, commands that require a second pass of the data).

When a CONTROLLED BY SourceFile is used, you should check the SourceFile SYS-IO-STATUS, to detect whether the access has been successful. Please refer to the section Procedural Commands (page 39) for returned values of the SYS-IO-STATUS.

Example 1: Controlling Database Access from an External File

Write a program that prints the order information for sales region 2 in the database for customers in the CUSTOMER-CONTROL SourceFile.

Assume that the CUSTOMER-CONTROL file contains CUSTOMER-NUMBER-CONTROL values, and that the CUSTSS01 subschema contains CUSTOMER, INVOICE, and ITEM records. CUSTOMER is a CALC record whose storage-keyfield is CUSTOMER-NUMBER.

Program Code

```
FIELD WK-IDMSCUST TYPE CHARACTER SIZE 17 INITIAL 'CUSTSS01'
SOURCEFILE CUSTOMER-CONTROL
SOURCEFILE IDMSCUST SCHEMA CUSTSCHM VERSION 1
      DBNAME WK-IDMSCUST
      CONTROLLED BY CUSTOMER-CONTROL
      KEY = CUSTOMER-NUMBER-CONTROL
      PATH (CUSTOMER,INVOICE VIA CUST-INVOICE,
            ITEM VIA INVOICE-ITEM)
REPORT 1
.
.
BEGIN RECORD CUSTOMER INPUT
IF SALES-REGION NE 2 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

MetaSuite reads the CUSTOMER-CONTROL SourceFile sequentially, and retrieves each CUSTOMER record directly, with its related records. It uses the value of CUSTOMER-NUMBER-CONTROL (from the CUSTOMER-CONTROL SourceFile) as the CALC key for the CUSTOMER record.

The CUSTOMER-CONTROL records can be in any sequence; they need not be sorted.

The record input procedure for the CUSTOMER record uses the *EXCLUDE* command to eliminate any CUSTOMER records from sales regions other than 2. No INVOICE records are retrieved for eliminated CUSTOMER records. All retrieved controlled sets are passed to report processing.

The report (or TargetFile) input procedure checks for the following conditions: a missing CUSTOMER record (meaning that no CUSTOMER exists for a particular CUSTOMER-CONTROL-KEY value); a CUSTOMER record that owns no INVOICE records; and an INVOICE record that owns no ITEM records.

The controlled sets available for report processing might be:

```
CUSTOMER-NUMBER
CONTROL          CUSTOMER     INVOICE     ITEM
1620921286       1620921286   SC20221     CCC11111
                                          DDD22221
                                          DDD22225
                              SC20344     CCC11233
        .     .      .     .
```

All INVOICE and ITEM data for CUSTOMER 1620921286. If there are duplicate CUSTOMER records (with the same key value), they would follow with their INVOICE and ITEM records.

```
        .     .      .     .
2248374765       2248374765   SC41532     CCC22244
        .     .      .     .
```

Example 2: Controlling Database Access from within the Database Itself

This example shows how to use the CONTROLLED BY option to retrieve database records, based on CALC-key values stored in another record type in the same database.

Problem Statement

Produce a report that lists all customers and their invoices, along with the name of the salesperson that wrote each invoice. Only process invoices written by salespeople in sales region 3.

Assume that the INVOICE record in the CUSTSS01 subschema has a field called INVOICE-WRITTEN-BY, which is the CALC key of an associated SALESPERSON record defined in the same subschema.

Program Code

Use two SOURCEFILE commands for the CUSTSS01 subschema: one to access CUSTOMER and INVOICE records, the other to access only SALESPERSON records. Use the PREFIX option to clarify which entities are referenced by which FILE command.

```
FIELD WK-IDMSCUST TYPE CHARACTER SIZE 17 INITIAL 'CUSTSS01'
FIELD WK-WRT-IDMSCUST TYPE CHARACTER SIZE 17 INITIAL 'CUSTSS01'
SOURCEFILE IDMSCUST SCHEMA CUSTSCHM VERSION 1
      DBNAME WK-IDMSCUST
      PATH (CUSTOMER,INVOICE VIA CUST-INVOICE)
SOURCEFILE IDMSCUST PREFIX 'WRT-' SCHEMA CUSTSCHM
      VERSION 1 DBNAME WK-WRT-IDMSCUST
      CONTROLLED BY IDMSCUST KEY = ORDER-WRITTEN-BY
      PATH (WRT-SALESPERSON)
REPORT 1
.
.
BEGIN RECORD CUSTOMER INPUT
IF WRT-SALES-REGION NE 3 EXCLUDE
BEGIN REPORT 1 INPUT
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 CUSTOMER record input procedure excludes records for regions other than 3. MetaSuite does not build controlled sets for the unwanted sales regions.

The report input procedure checks for the following conditions: a missing INVOICE record (that is, a customer with no stored invoice information), and a missing SALESPERSON record (which means that the ORDER-WRITTEN-BY information is incorrect).

The procedural code is the same as if there were a link between INVOICE and SALESPERSON (and they were included together in the PATH specification)

6.4. Procedural Commands

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

Checking the Return Status

To check the status information returned to MetaSuite by CA-IDMS or returned by a MetaSuite DML command, reference the following system fields:

This System field ...	Contains the ...
SourceFile-name SYS-INTERNAL-STATUS	CA-IDMS status code returned from the last request to CA-IDMS, using the subschema specified by the DBNAME of the SourceFile.
SourceFile-name SYS-IO-STATUS	I/O status code that you can check using one of the following constants: SYS-OK indicates that a CA-IDMS status code of "0000" was returned, and that the input record was successfully validated by MetaSuite. SYS-NOT-RELATED indicates that the IF MEMBER command returns that the current record is not a member of the set. SYS-ERROR indicates that a non-zero CA-IDMS status code was returned, or that an input validation failed for a successfully obtained record. In the case of a validation error, the SourceFile-name SYS-INTERNAL-STATUS contains zeroes, and the data in the record fields is unpredictable. SYS-EOF indicates that a HALT SOURCEFILE command has been executed for the named SourceFile.
SYS-RECORD	Name of the record most recently obtained. When you use the MetaSuite DML commands to obtain a member record in a multi-member set using a form of the IDMS OBTAIN command that does not specify a record name, reference SYS-RECORD for the name of the record obtained.

6.5. Command Summary

The MetaSuite commands used to access a CA-IDMS 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.
HALT ALL	Stop all processing.
HALT SOURCEFILE	Stop processing of one or more SourceFiles.
ACCEPT FROM CURRENCY	Return the Db-key for the current record.
ACCEPT FROM OWNER CURRENCY	Return the Db-key of the owner record, related to the current record.
RELEASE	Release data to the intermediate sort or extract SourceFile (in a SourceFile initial procedure), or to the report (or TargetFile) processing logic (in a SourceFile input procedure).
START	Position a SourceFile at a particular record before beginning access.

6.6. EXCLUDE

The following describes the EXCLUDE command and its use in accessing CA-IDMS databases.

Command Syntax

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

Usage

The EXCLUDE command is used within a procedure, to bypass processing of the current record, or any specified record of the SourceFile. This command operates a described in the *MetaMap Manager User Guide*, except as noted below.

For a non-MANUAL SourceFile (that is, SOURCEFILE statement does not include the MANUAL option), when an EXCLUDE command is processed within a SourceFile initial procedure for a SORT, EXTRACT or PREPASS, or within a SourceFile input procedure, the procedure is exited (and the record is bypassed). If the SourceFile has been HALTed, MetaSuite terminates the processing for the SourceFile. If the SourceFile has not been HALTed, MetaSuite executes the path-building code for the SourceFile, then re-executes the procedure, beginning with the first command. When the end-of-file condition is obtained, MetaSuite terminates processing for the SourceFile.

For a MANUAL subschema, when an EXCLUDE command is processed within a SourceFile initial procedure for a SORT, EXTRACT or PREPASS, or within a SourceFile input procedure where no SourceFile initial procedure for a SORT, EXTRACT or PREPASS has been previously defined, the procedure is exited. If the SourceFile has been HALTed, MetaSuite terminates the processing for the SourceFile. If the SourceFile has not been HALTed, MetaSuite re-executes the procedure, beginning with the first command.

Note that an EXCLUDE command within a SourceFile input procedure, that follows a SourceFile initial procedure for a SORT or EXTRACT, performs the same as for a SourceFile input procedure for a non-MANUAL subschema.

When an EXCLUDE command is processed within a record input procedure for a MANUAL SourceFile, MetaSuite exits the procedure.

Bypassing Processing for a Record

The EXCLUDE command is most useful in a record input procedure for a multi-record SourceFile path. In this case, you can use the EXCLUDE command, without the *record-name* or SourceFile-name options, to bypass processing for a record before any lower-level records are read into the path. Processing time is improved, because the overhead of constructing unwanted paths of records is eliminated.

We recommend that you use the EXCLUDE command in a record input procedure any time you want to bypass processing for a record based on information in that record or a higher-level record in the path.

For example, assume that a program contains the following commands:

```
FIELD WK-IDMSCUST TYPE CHARACTER SIZE 17 INITIAL 'CUSTSS01'
SOURCEFILE IDMSCUST SCHEMA CUSTSCHM VERSION 1
      DBNAME WK-IDMSCUST
      PATH (CUSTOMER,INVOICE VIA CUST-INVOICE)
REPORT 1
DETAIL 1 (CUST-NUMBER SHORT,INVOICE-NUMBER)
```

The following report might be produced. The path number for each detail line is shown to the right.

```
CUST          INVOICE
NUMBER        NUMBER         (path number)
*********     *******
162092128     SC20221        (1)
              SC20344        (2)
              SC39374        (3)
207384949     SC49483        (4)
              SC25342        (5)
              SC47365        (6)
299430123     SC36254        (7)
              SC41092        (8)
303300928     SC20982        (9)
```

Nine paths are constructed from the CA-IDMS database records accessed.

To bypass any customer whose account number begins with the character "2", you would add the following record input procedure to the program:

```
BEGIN RECORD CUSTOMER INPUT
IF CUST-NUMBER IR ('2000000000' TO '2999999999') -
       EXCLUDE
```

The report generated would contain only the information shown above in paths 1, 2, 3, and 9. MetaSuite would not read the INVOICE records shown in paths 4 through 8, reducing the number of database access requests by five.

If the decision to bypass processing for a record is based on information in other records in a path, use the EXCLUDE command in a SourceFile initial or SourceFile input procedure, as described next.

Bypassing Paths of Records

`record-name`

Optional and applicable only in SourceFile initial or SourceFile input procedures. The *record-name* option is used when a PATH of records is being accessed, to identify the specific record type in the path in which you are no longer interested. Record-name must be the name of a record specified in the PATH specification for the SourceFile.

The system skips occurrences of any other record types until the next occurrence of the named (excluded) record is encountered. Processing time is improved, because the overhead of constructing unwanted paths of records is eliminated.

For example, assume that a program contains the same SOURCEFILE, REPORT, and DETAIL commands shown above, under "Bypassing Processing for a Record". Also assume that the same report shown above, with path numbers indicated, might be produced.

To bypass only customers that have an account number beginning with the character "2" and an invoice whose invoice number begins with the characters "SC2", you would add the following SourceFile input procedure to the program:

```
BEGIN SOURCEFILE IDMSCUST INPUT
IF CUST-NUMBER IR ('2000000000' TO '2999999999') -
       AND INVOICE-NUMBER IR ('SC20000' TO 'SC29999') -
       EXCLUDE CUSTOMER
```

The report generated would contain only the information shown above in paths 1, 2, 3, 7, 8, and 9. MetaSuite would build paths 4, 5, and 6, but exclude them from processing.

Bypassing CONTROLLED BY Records

`SourceFile-name`

Optional, and applicable only on a CONTROLLED BY SourceFile and only in SourceFile input procedures. The SourceFile-name option is used to identify the controlling SourceFile data in which you are not interested. SourceFile-name is the name of a controlling SourceFile (defined through the CONTROLLED BY option of the SOURCEFILE command).

For example, assume that a program contained the following code:

```
FIELD WK-IDMSCUST TYPE CHARACTER SIZE 17 INITIAL 'CUSTSS01'
FIELD WK-WRT-IDMSCUST TYPE CHARACTER SIZE 17 INITIAL 'CUSTSS01'
SOURCEFILE IDMSCUST SCHEMA CUSTSCHM VERSION 1
      DBNAME WK-IDMSCUST
      PATH (CUSTOMER,INVOICE VIA CUST-INVOICE)
SOURCEFILE IDMSCUST PREFIX 'WRT-' SCHEMA CUSTSCHM
      VERSION 1 DBNAME WK-WRT-IDMSCUST
      PATH (WRT-SALESPERSON)
      CONTROLLED BY IDMSCUST KEY = ORDER-WRITTEN-BY
REPORT 1
DETAIL 1 (CUSTOMER-NUMBER SHORT,INVOICE-NUMBER, -
      WRT-SALES-REGION,WRT-SALESPERSON)
```

The following report might be produced. The controlled set number for each detail line is shown to the right.

```
CUST          INVOICE        WRT
NUMBER        NUMBER         SALESPERSON    (set number)
*********     *******        ***********
162092128     SC20221         JONES         (1)
              SC20344         BLACK         (2)
              SC39374         REYES         (3)
```

```
207384949     SC49483         CHANG         (4)
              SC25342         SMITH         (5)
              SC47365         KELLY         (6)
299430123     SC36254         GABLE         (7)
              SC41092         HERON         (8)
303300928     SC20982         HAMON         (9)
```

Nine controlled sets were constructed from the CA-IDMS database records accessed.

If you wanted to process salespersons from the Southwest region only, you could add the following SourceFile input procedure code:

```
BEGIN SOURCEFILE WRT-IDMSCUST INPUT
IF WRT-SALES-REGION NE 3 -
      EXCLUDE IDMSCUST
```

This procedure checks the sales-region value in the SALESPERSON record, and if it is not the desired regions, excludes the current records from the higher-level IDMSCUST SourceFile from processing. MetaSuite retrieves the next INVOICE record, which in turn causes the lower-level (WRT-IDMSCUST) SourceFile processing to retrieve a new SALESPERSON record. The report output would include only the sales persons from region 3.

6.7. EXIT

The following describes the *EXIT* command and its use in accessing CA-IDMS databases.

Command Syntax

```
EXIT
```

Usage

The EXIT command leaves the current procedure and returns to the "calling" procedure. This command operates as described in the *MetaMap Manager User Guide*, except as detailed below.

For a MANUAL SourceFile, EXIT processing is the same as for a non-MANUAL SourceFile, except that:

- If the SourceFile initial procedure defines a SORT or EXTRACT, and it contains:
  - No RELEASE command, MetaSuite releases a record to the sort or extract SourceFile that contains the current values of all referenced fields from the MANUAL SourceFile, as well as any necessary sort-key values, when it executes an EXIT command.
  - One or more RELEASE commands, MetaSuite does not release a record to the sort or extract SourceFile when it executes an EXIT command.

  If processing for the SourceFile has not been HALTed, processing continues at the first command of the SourceFile initial procedure. If processing for the SourceFile has been HALTed, processing terminates.
- If the SourceFile initial procedure does not include a SORT or EXTRACT command, and the SourceFile input procedure includes an EXIT command and it contains:
  - No RELEASE command, MetaSuite executes the report input logic when it executes an EXIT command.
  - One or more RELEASE commands, MetaSuite does not execute the report input logic when it executes the EXIT command.

  If processing for the SourceFile has not been HALTed, processing continues at the first command of the SourceFile input procedure. If processing for the SourceFile has been HALTed, processing terminates.

6.8. GET

The following describes the *GET* command and its use in accessing CA-IDMS databases.

Command Syntax

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

Usage

The GET command is used to read records in a CA-IDMS database from within procedural code.

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

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 PATH option.

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, it must be a CALC or indexed record. If the GET command names a SourceFile, the entry record in the SourceFilePath must be a CALC or 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 CALC-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 SOURCEFILE and GET command options specified for the CA-IDMS 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 a CALC Record

The following code uses the GET command to retrieve a CALC record, using a CALC-key value:

```
FIELD WK-IDMSCUST TYPE CHARACTER SIZE 17 INITIAL 'CUSTSS01'
SOURCEFILE IDMSCUST SCHEMA CUSTSCHM VERSION 1
      DBNAME WK-IDMSCUST
      CONTROLLED
      PATH (CUSTOMER)
      .
      .
      .
GET CUSTOMER KEY = '2903837698'
IF IDMSCUST SYS-IO-STATUS EQ SYS-ERROR -
      CUST-NAME = 'NOT FOUND'
```

The CUSTOMER record was defined to the dictionary with CUST-NUMBER, a 10-character alphanumeric field, specified as the storage-keyfield.

If the CUSTOMER record with the desired storage-key is not found in the database, the name prints as 'NOT FOUND'.

Example 2: Retrieving an Indexed Record

The following code uses the GET command to retrieve a CUSTOMER record, using the index IX-CUST-NAME:

```
FIELD WK-IDMSCUST TYPE CHARACTER SIZE 17 INITIAL 'CUSTSS01'
SOURCEFILE IDMSCUST SCHEMA CUSTSCHM VERSION 1
      DBNAME WK-IDMSCUST
      PATH (CUSTOMER VIA IX-CUST-NAME)
      .
      .
GET CUSTOMER KEY = 'HUDSON RIVER SPRINGS'
IF IDMSCUST SYS-INTERNAL-STATUS EQ '0326' -
      CUST-NAME = 'NOT FOUND'
```

If the CUSTOMER record is not found, 'NOT FOUND' appears in the name field.

Example 3: Retrieving a Path of Records

The following code uses the GET command to retrieve a path of records for the IDMSCUST SourceFile:

```
FIELD WK-IDMSCUST TYPE CHARACTER SIZE 17 INITIAL 'CUSTSS01'
SOURCEFILE IDMSCUST SCHEMA CUSTSCHM VERSION 1
      DBNAME WK-IDMSCUST
      CONTROLLED
      PATH (CUSTOMER, INVOICE VIA CUST-INVOICE,
            ITEM VIA INVOICE-ITEM OCCURS 45)
      .
      .
GET CUSTOMER KEY = '2903837698'
IF IDMSCUST SYS-IO-STATUS EQ SYS-ERROR -
      CUST-NAME = 'NOT FOUND'
```

This GET command retrieves the same entry record as the first example, as well as the first INVOICE record for that CUSTOMER and up to 45 ITEM records (beginning with the first record) for that INVOICE. The number of ITEM records actually retrieved is in the system field, ITEM SYS-PATH-COUNT, following the execution of each GET command.

6.9. HALT ALL

The following describes the *HALT ALL* command and its use in accessing CA-IDMS databases.

Command Syntax

```
HALT ALL
```

Usage

The HALT ALL command stops all processing. This command operates identically to the HALT ALL command described in the *MetaMap Manager User Guide*, except that it also issues a FINISH request to the CA-IDMS database system for all subschemas defined by the SourceFiles currently in use.

6.10. HALT SOURCEFILE

The following describes the *HALT SOURCEFILE* command and its use in accessing CA-IDMS databases.

Command Syntax

```
HALT SOURCEFILE [(SourceFile-name,...)
```

Usage

The HALT SOURCEFILE command halts the processing of one or more SourceFiles. This command operates identically to the HALT SOURCEFILE command described in the *MetaMap Manager User Guide*, except that MetaSuite issues a FINISH request to CA-IDMS for the subschemas defined by the SourceFiles.

Identifying the SourceFile(s) to Be Halted

`(SourceFile-name,...)`

Optional. *SourceFile-name* identifies the SourceFile whose processing is being halted. It must specify a SourceFile that does not include the CONTROLLED option.

If SourceFile-name is omitted from the command within a SourceFile procedure, it defaults to the SourceFile within whose (SourceFile Initial or SourceFile Input) procedure the command is contained.

6.11. ACCEPT FROM CURRENCY

The following describes the IDMS DML *ACCEPT FROM CURRENCY* command and its use in accessing CA-IDMS databases.

Command Syntax

```
EXEC-IDMS
      ACCEPT field-name FROM record-name CURRENCY
END-EXEC
```

Usage

This CA-IDMS DML Command can be used on Automatic SourceFiles to obtain the DB-Key of the current record being processed. Please refer to MetaSuite CA-IDMS DML Commands (page 49) for further usage.

6.12. IDMS ACCEPT FROM SET

The following describes the IDMS DML *ACCEPT FROM SET* command and its use in accessing CA-IDMS databases.

Command Syntax

```
EXEC-IDMS
      ACCEPT field-name FROM set-name OWNER CURRENCY
END-EXEC
```

Usage

This CA-IDMS DML Command can be used on Automatic SourceFiles to obtain the DB-Key of the owner within the specified set of the current record being processed. Please refer to the section MetaSuite CA-IDMS DML Commands (page 49) for further usage.

6.13. RELEASE

The following describes the *RELEASE* command and its use in accessing CA-IDMS databases.

Command Syntax

```
RELEASE
```

Usage

Please refer to MetaSuite CA-IDMS DML Commands (page 49) for usage.

6.14. START

The following describes the *START* command and its use in accessing CA-IDMS 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 PATH option may be specified. The entry record 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.

CHAPTER 7

MetaSuite CA-IDMS DML Commands

7.1. Overview

The MetaSuite DML commands, each enclosed between the keywords EXEC-IDMS and END-EXEC (translated to IDMS in the MSL, MetaSuite Specification Language), allow you to access the database using syntax that closely parallels CA-IDMS DML commands.

You can use the MetaSuite DML commands, in any type of procedure, for any CA-IDMS SourceFile that includes the MANUAL specification. An exception is if the SourceFile initial procedure for a MANUAL SourceFile contains the SORT or EXTRACT command. In this case, you can use these commands only within that SourceFile initial procedure.

When using the CA-IDMS OBTAIN commands, be aware of the following considerations:

- Your procedure logic must ensure the appropriate currencies within the database.

If you code a CA-IDMS OBTAIN command for a SourceFile whose MANUAL specification does not include the CONTROLLED option, you must include a HALT command at the appropriate processing point. If you do not include a HALT command, a processing loop can occur. MetaSuite does not verify that the HALT command is coded in the program.

7.2. Checking the Return Status

To check the status information returned by a MetaSuite DML command, reference the SourceFile-name SYS-INTERNAL-STATUS, SourceFile-name SYS-IO-STATUS, and SYS-RECORD system fields, as described in the previous chapter.

7.3. Command Summary

The MetaSuite DML commands used to access a CA-IDMS database are summarized below, then discussed individually:

This Command ...	Is Used to ...
ACCEPT field-name FROM CURRENCY	Return the Db-key for the current record of a specified type or the current record in a specified set.
ACCEPT field-name FROM set-name	Return the Db-key of the next, prior, or owner record, related to the current record of a specified set.

This Command ...	Is Used to ...
IF	Test for the presence of member records in a set, and determine whether a record is a member of a set.
OBTAIN DB KEY	Obtain a record directly, using a Db-key value.
OBTAIN CURRENCY	Obtain the current record of a specified type or within a specified set.
OBTAIN WITHIN set	Obtain a record in a named set.
OBTAIN WITHIN area	Obtain a record in a named area.
OBTAIN OWNER	Obtain the owner record of a named set.
OBTAIN CALC/DUPLICATE	Obtain a CALC record, using its CALC-key value.
OBTAIN WITHIN set USING sort-key	Obtain a record in a sorted set, using its sort-key value.
RELEASE	Release data to the intermediate sort or extract SourceFile (in a SourceFile initial procedure), or to the report (or TargetFile) processing logic (in a SourceFile input procedure).

In addition, note discussion in the previous chapter regarding the use of the following commands with a MANUAL SourceFile:

- EXCLUDE
- EXIT

7.4. ACCEPT FROM CURRENCY

The following describes the *ACCEPT FROM CURRENCY* command and its use in accessing CA-IDMS databases.

Command Syntax

```
EXEC-IDMS
      ACCEPT field-name FROM {CURRENCY |
record-name CURRENCY |
set-name CURRENCY}
END-EXEC
```

Usage

This form of the ACCEPT command returns the Db-key for the current record for a SourceFile, or the current record of a specified record type.

If the program accesses the database using multiple SourceFiles whose SOURCEFILE commands include the MANUAL or CONTROLLED MANUAL option, it is recommended that you include the record-name or set-name specification here.

Identifying the GlobalField for the Db-key

field-name

Required. *Field-name* is the name of the 4-byte binary GlobalField in which you want the requested Db-key returned.

Requesting the Db-key of the Current Record

`CURRENCY`

Optional. The CURRENCY option returns the Db-key of the current record for a SourceFile. The Db-key returned depends on where the command is coded. If the command appears within:

- Any SourceFile procedure whose SOURCEFILE command includes the MANUAL option, the system returns the Db-key for the current record for the SourceFile.
- Any other procedure type, the system determines the first SOURCEFILE command that includes the MANUAL or CONTROLLED MANUAL option, and returns the Db-key of the current record for that SourceFile.

Specifying a Record Name

`record-name` `CURRENCY`

Optional. This option returns the Db-key of the current record of the type specified by record-name. If the PREFIX option was specified on the SOURCEFILE command for the SourceFile that includes the record type, you must include the prefix in the record name.

For example, to request that the Db-key of the current CUSTOMER record be returned in the field CUST-DB-KEY, you would code:

```
EXEC-IDMS ACCEPT CUST-DB-KEY FROM CUSTOMER CURRENCY END-EXEC
```

Specifying a Set Name

`set-name` `CURRENCY`

Optional. The option returns the Db-key of the current record for the set specified by set-name. If the PREFIX option was specified on the FILE command for the subschema that includes the set, you must include the prefix in the set name.

For example, to request that the Db-key of the current record for the CUST-SALES set be returned in the field CUST-SALES-KEY, you would code:

```
EXEC-IDMS ACCEPT CUST-SALES-KEY FROM CUST-SALES CURRENCY END-EXEC
```

7.5. ACCEPT FROM SET

The following describes the *ACCEPT FROM SET* command and its use in accessing CA-IDMS databases.

Command Syntax

```
EXEC-IDMS
      ACCEPT field-name FROM set-name
      {NEXT | PRIOR | OWNER} CURRENCY
END-EXEC
```

Usage

This form of the ACCEPT command returns the Db-key of the next, prior, or owner record, relative to the current record of a specified set.

Identifying the GlobalField for the Db-key

field-name

Required. *Field-name* is the name of the 4-byte binary GlobalField in which you want the requested Db-key returned.

Identifying the Set

FROM *set-name*

Required. *Set-name* is the name of the set that you want processed. If the PREFIX option was specified on the SOURCEFILE command for the SourceFile that includes the set, you must include the prefix in the set name.

Specifying the Record

{NEXT | PRIOR | OWNER} CURRENCY

Required. NEXT returns the Db-key for the next record in the set specified by set-name. PRIOR returns the Db-key for the prior record in the specified set. OWNER returns the Db-key for the owner record in the specified set.

7.6. IF MEMBER

The following describes the *IF MEMBER* command and its use in accessing CA-IDMS databases.

Command Syntax

```
EXEC-IDMS
      IF {[NOT] set-name MEMBER |
set-name IS [NOT] EMPTY}
END-EXEC
```

Usage

The IF command is used for either of the following purposes:

- To determine whether a record is a member of an optional set.
- To determine whether there are member records in an optional set.

To check the results of an IF command, use the MetaSuite status code returned by the command in SourceFile-name SYS-IO-STATUS or the CA-IDMS status code returned in SourceFile-name SYS-INTERNAL-STATUS, as described for each command option. For information about CA-IDMS status codes, see the CA-IDMS Programmer's Reference Guide - COBOL, " Computer Associates.

Testing a Record for Set Membership

`[NOT] set-name MEMBER`

Optional. This option determines whether or not the current record for a SourceFile is a member of the set specified by set-name. The set named must be an optional set. If the PREFIX option was specified on the SOURCEFILE command for the SourceFile that includes the set, you must include the prefix in the set name.

The optional NOT specification is available for documentation purposes only. The processing for the command is unaffected by inclusion of the NOT option.

The command returns a status code as follows:

SourceFile-name SYS-IO-STATUS	SourceFile-Name SYS-INTERNAL-STATUS	Meaning
SYS-OK	0000	Record is a member of the set.
SYS-NOT-RELATED	1601	Record is not a member of the set.
SYS-ERROR	1606, 1608, or 1613	An error has occurred

For example, to test whether the current record is a member of the INVOICE-IREMARK set, code:

```
EXEC-IDMS IF INVOICE-IREMARK MEMBER END-EXEC
```

Testing a Set for Member Records

`set-name IS [NOT] EMPTY`

Optional. This option determines whether or not the current owner of the set specified by set-name owns any member records (that is, whether or not the set is empty). The set named must be an optional set. If the PREFIX option was specified on the SOURCEFILE command for the SourceFile that includes the set, you must include the prefix in the set name.

The optional NOT specification is available for documentation purposes only. The processing for the command is unaffected by inclusion of the NOT option.

The command returns a status code as follows:

SourceFile-name SYS-IO-STATUS	SourceFile-name SYS-INTERNAL-STATUS	Meaning
SYS-OK	0000	Set is empty.
SYS-ERROR	1601, 1606, 1608, or 1613	Set is not empty.

For example, to test whether the current INVOICE record owns any member IREMARK records in the INVOICE-IREMARK set, code:

```
EXEC-IDMS IF INVOICE-IREMARK IS EMPTY END-EXEC
```

7.7. OBTAIN DB-KEY IS

The following describes the *OBTAIN DB-KEY* IS command and its use in accessing CA-IDMS databases.

Command Syntax

```
EXEC-IDMS
      OBTAIN record-name DB-KEY IS field-name
END-EXEC
```

Usage

This for of the OBTAIN command obtains a record directly, using a Db-key value.

Specifying a Record Name

record-name

Required. *Record-name* identifies the type of record you want to obtain. The record named must be included in a SourceFile whose SOURCEFILE command specifies the CONTROLLED MANUAL options. If the PREFIX option was specified on the SOURCEFILE command, you must include the prefix in the record name.

Identifying the GlobalField for the Db-key

DB-KEY IS *field-name*

Required. *Field-name* is the name of the field that contains the Db-key of the record you want to obtain. The field named must be a 4-byte binary GlobalField.

7.8. OBTAIN CURRENT WITHIN SET

The following describes the *OBTAIN CURRENT WITHIN SET* command and its use in accessing CA-IDMS databases.

Command Syntax

```
EXEC-IDMS
      OBTAIN CURRENT [record-name | WITHIN set-name]
END-EXEC
```

Usage

This form of the OBTAIN command obtains the current record for a SourceFile, or the current record of a specified record type or a specified set.

If the program accesses the database using multiple SourceFiles whose SOURCEFILE commands include the CONTROLLED MANUAL option, it is recommended that you include the *record-name* or WITHIN *set-name* option in the command.

Requesting the Current Record for a Subschema

Without the record-name or WITHIN set-name options, the command obtains the current record for a SourceFile. The record obtained depends on where the command is coded. If the command appears within:

- Any SourceFile procedure for a SourceFile whose SOURCEFILE command includes the MANUAL option, the system returns the current record for the SourceFile.
- Any other procedure type, the system determines the first SOURCEFILE command for a SourceFile that includes the CONTROLLED MANUAL option, and returns the current record for that SourceFile.

Specifying a Record Name

`record-name`

Optional. This option returns the current record of the type specified by record-name. If the PREFIX option was specified on the SOURCEFILE command for the SourceFile that includes the record type, you must include the prefix in the record name.

For example, to request that the current CUSTOMER record be obtained, code:

```
EXEC-IDMS OBTAIN CURRENT CUSTOMER END-EXEC
```

Specifying a Set Name

`WITHIN set-name`

Optional. This option returns the current record for the set specified by set-name. If the PREFIX option was specified on the SOURCEFILE command for the SourceFile that includes the set, you must include the prefix in the set name.

For example, to request that the current record for the CUST-SALES set be obtained, code:

```
EXEC-IDMS OBTAIN CURRENT WITHIN CUST-SALES END-EXEC
```

7.9. OBTAIN WITHIN SET

The following describes the *OBTAIN WITHIN SET* command and its use in accessing CA-IDMS databases.

Command Syntax

```
EXEC-IDMS
      OBTAIN {NEXT | PRIOR | LAST | FIRST | count-field}
     [record-name] WITHIN set-name
END-EXEC
```

Usage

This form of the OBTAIN command obtains a record, relative to the current record in a named set.

Specifying the Relative Record to Obtain

`{NEXT | PRIOR | LAST | FIRST | count-field}`

Required. These specifications determine the record to be obtained, relative to the current record of the set. NEXT, PRIOR, FIRST, and LAST obtain the next, prior, first, and last record in the set, respectively.

Count-field obtains the nth record in the set, relative to the current record, using the number contained in the count field. Count-field must be a numeric field that contains a non-zero whole number. If the number is positive, the system obtains the nth record in the "next" direction. If the number is negative, the system obtains the nth record in the "prior" direction. If count-field contains a negative number, the set must contain prior pointers.

Specifying a Record Name

record-name

Optional. *Record-name* identifies the type of record you want to obtain. If the PREFIX option was specified on the SOURCEFILE command for the SourceFile that includes the record, you must include the prefix in the record name.

Without the record-name option, the command obtains the next record in the set, regardless of its type. The SYS-RECORD system GlobalField contains the name of the record obtained.

Specifying the Set

WITHIN *set-name*

Required. *Set-name* is the name of the set from which you want to obtain a record. If the PREFIX option was specified on the SOURCEFILE command for the SourceFile that includes the set, you must include the prefix in the set name.

7.10. OBTAIN WITHIN AREA

The following describes the *OBTAIN WITHIN AREA* command and its use in accessing CA-IDMS databases.

Command Syntax

```
EXEC-IDMS
      OBTAIN {NEXT | PRIOR | LAST | FIRST | count-field}
record-name WITHIN area-name
END-EXEC
```

Usage

This form of the OBTAIN command obtains a record, relative to the current record in a named area.

Specifying the Relative Record to Obtain

{NEXT | PRIOR | LAST | FIRST | *count-field*}

Required. These specifications determine the record to be obtained, relative to the current record of a specified type in the area. NEXT, PRIOR, FIRST, and LAST obtain the next, prior, first, and last record, respectively, of the type specified by record-name.

Count-field obtains the nth record of the specified type, relative to the current specified record, using the number contained in the count field. Count-field must be a numeric field that contains a non-zero whole number. If the number is positive, the system obtains the nth record in the "next" direction. If the number is negative, the system obtains the nth record in the "prior" direction. If count-field contains a negative number, the set must contain prior pointers.

Specifying a Record Name

record-name

Required. *Record-name* identifies the type of record you want to obtain. If the PREFIX option was specified on the SOURCEFILE command for the SourceFile that includes the record, you must include the prefix in the record name.

Specifying the Area

WITHIN *area-name*

Required. *Area-name* is the name of the area from which you want to obtain a record. If the PREFIX option was specified on the SOURCEFILE command for the SourceFile that includes the record, you must include the prefix in the area name.

7.11. OBTAIN OWNER WITHIN SET

The following describes the *OBTAIN OWNER WITHIN SET* command and its use in accessing CA-IDMS databases.

Command Syntax

```
EXEC-IDMS
      OBTAIN OWNER WITHIN set-name
END-EXEC
```

Usage

This form of the OBTAIN command obtains the owner of the current occurrence of a specified set.

Specifying a Set Name

set-name

Required. *Set-name* identifies the set for which you want to obtain the owner. If the PREFIX option was specified on the SOURCEFILE command for the SourceFile that owns the set, you must include the prefix in the set name.

7.12. OBTAIN RECORD-NAME

The following describes the *OBTAIN RECORD-NAME* command and its use in accessing CA-IDMS databases.

Command Syntax

```
EXEC-IDMS
      OBTAIN {CALC | ANY | DUPLICATE} record-name
END-EXEC
```

Usage

This form of the OBTAIN command obtains a CALC record, using its CALC-key value. You must have previously assigned the CALC-key value to the appropriate storage-keyfield for the record.

Specifying Which Record to Obtain

`{CALC | ANY | DUPLICATE}`

Required. These options determine the record to be obtained. CALC and ANY are synonymous and obtain the first occurrence of the record type whose CALC key matches the value in the storage-keyfield for the record type.

UPLICATE obtains the next record with the same CALC-key value as the current record of the specified type. To use the DUPLICATE option, you must first obtain a record with the same CALC-key value, using the CALC or ANY option.

Specifying a Record Name

`record-name`

Required. Record-name identifies the type of CALC record you want to obtain. If the PREFIX option was specified on the SOURCEFILE command for the SourceFile that includes the record type, you must include the prefix in the record name.

7.13. OBTAIN WITHIN SET USING SORT KEY

The following describes the OBTAIN WITHIN SET USING SORT KEY command and its use in accessing CA-IDMS databases.

Command Syntax

```
EXEC-IDMS
      OBTAIN record-name WITHIN set-name [CURRENT]
      USING sort-key
END-EXEC
```

Usage

This form of the OBTAIN command obtains a record in a sorted set, using its sort-key value.

Specifying a Record Name

record-name

Required. *Record-name* identifies the type of record you want to obtain. If the PREFIX option was specified on the SOURCEFILE command for the SourceFile that includes the record type, you must include the prefix in the record name.

Specifying a Set Name

WITHIN *set-name* [CURRENT]

Required. *Set-name is* the name of the set that you want to process. The set named must be a sorted set. If the PREFIX option was specified on the SOURCEFILE command for the SourceFile that owns the set, you must include the prefix in the set name.

Within the CURRENT option, the command begins the search for the record from the current record for the set. Without this option, the command begins the search with the current owner record for the set.

Specifying the Sort-Key Value

USING *sort-key*

Required. *Sort-key* is the sort-control field for the record. You must ensure that this field contains the sort-key value of the record to be obtained, before using the command. Be aware that MetaSuite cannot verify the named field as the sort-key field defined for the set.

If the PREFIX option was specified on the SOURCEFILE command for the SourceFile that owns the record and set, you must include the prefix in the sort-key name.

7.14. RELEASE

The following describes the *RELEASE* command and its use in accessing CA-IDMS databases.

Command Syntax

RELEASE

Usage

The RELEASE command releases data from a MANUAL SourceFile to the intermediate sort or extract SourceFile (in a SourceFile initial procedure), or to the report processing logic (in a SourceFile input procedure). In the latter case, the data is processed for each requested report, in report number sequence.

You must include this command within either:

- A SourceFile initial procedure for a MANUAL SourceFile, or
- A SourceFile input procedure for a MANUAL SourceFile whose SourceFile initial procedure contains no SORT or EXTRACT commands.

If you include one or more RELEASE commands within a procedure, the system releases the file data to the intermediate file or report processing logic only when a RELEASE command is executed. If you do not include a RELEASE command within a procedure, MetaSuite automatically releases the data at the end of the procedure, or when an EXIT command is executed.

Appendix A - MetaStore Manager Collect for CA-IDMS

A.1. Overview

This chapter introduces the Collect File functionality of the MetaStore Manager to capture CA-IDMS definitions from IDD. You should be familiar with the CA-IDMS concepts presented in the section CA-IDMS Concepts and Terminology (page 4) before reading this chapter.

The relationships between the basic MetaSuite and CA-IDMS terms are as follows:

MetaSuite and CA-IDMS Terms

Metasuite	CA-IDMS
File	Schema
Record	Record Type
Field	Field
Index	CA-IDMS Index
Link	CA-IDMS set

A.2. IDMS SCHEMA

When starting the Collect File functionality of the MetaStore Manager, the option "IDMS Schema punch" represents a possibility to transform CA-IDMS IDD definitions into MetaSuite format.

Source information

You will first need to produce a Schema punch from the CA-IDMS IDD. This can be done by the following example JCL:

The input is an IDMS punch of your schema as a result of the following example JCL for z/OS:

```
//IDMSCHEM EXEC PGM=IDMSCHEM
//SYSCTL DD DSN=IDMS-SYSCTL-filename,DISP=SHR
//SYSUDUMP DD SYSOUT=R
//SYSLST DD SYSOUT=R
//SYSPCH DD SYSOUT=R
//SYSIPT DD *
SIGNON … DICTNAME IS IDMS-DatabaseName
PUNCH SCHEMA NAME IS Schemaname VERSION HIGH
WITH AREAS
ALSO WITH RECORDS
ALSO WITH ELEMENTS
ALSO WITH DETAILS
ALSO WITH SETS
AS SYNTAX
```

The result of this JCL will need to be transferred to your workstation, so that it is available as source for MetaStore Manager.

IDMS File Information

When starting the "IDMS Schema" collect option, you will need to specify the filename of the result of the PUNCH IDMS.

During the capture of file definition, IDMS File Information will be asked. It consists of 3 types of information:

- File Name
- Subschema Name
- Database Name

File Name

The Filename of the IDMS file as it will be stored in the MetaStore.

Subschema Name

The subschema name that will be accessed by default by the MetaMap Manager programs working on this schema.

Database Name

The CA-IDMS database name that will be accessed by default by the MetaMap Manager programs working on this schema.

Result

A dictionary file will be produced which represents the correct schema definition as stored in IDD.

Example

Consider the following CA-IDMS PUNCH for the schema IDMSSCHM:

```
ADD
SCHEMA NAME IS CUSTSCHM VERSION IS 1
MEMO DATE IS 30/07/90
ASSIGN RECORD IDS FROM 1001
.
ADD
AREA NAME IS CUST-AREA
ESTIMATED PAGES ARE 0
.
ADD
RECORD NAME IS CUSTOMER
RECORD ID IS 0611
LOCATION MODE IS CALC USING (CUST-NUMBER )
      DUPLICATES ARE NOT ALLOWED
CALL W02A0001 BEFORE STORE
CALL W02A0001 BEFORE MODIFY
WITHIN AREA CUST-AREA OFFSET 0 PERCENT FOR 100 PERCENT
.
02 CUST-NUMBER
      PICTURE IS X(10)
      USAGE IS DISPLAY
.
02 CUST-NAME
      PICTURE IS X(20)
      USAGE IS DISPLAY
.
02 CUST-CITY
      PICTURE IS X(15)
      USAGE IS DISPLAY
.
ADD
RECORD NAME IS INVOICE
RECORD ID IS 0620
LOCATION MODE IS CALC USING (INVOICE-NUMBER)
CALL W02A0001 BEFORE STORE
CALL W02A0001 BEFORE MODIFY
WITHIN AREA CUST-AREA OFFSET 0 PERCENT FOR 100 PERCENT
.
02 INVOICE-NUMBER
PICTURE IS X(10)
USAGE IS DISPLAY
.
02 INVOICE-DATE
      PICTURE IS X(10)
      USAGE IS DISPLAY
.
ADD
```

```
RECORD NAME IS ITEM
RECORD ID IS 0621
LOCATION MODE IS VIA INVOICE-ITEM SET
CALL W02A0001 BEFORE STORE
CALL W02A0001 BEFORE MODIFY
WITHIN AREA CUST-AREA OFFSET 0 PERCENT FOR 100 PERCENT
.
02 PRODUCT-NUMBER
      PICTURE IS X(5)
      USAGE IS DISPLAY
.
02 PRODUCT-DESCRIPTION
      PICTURE IS X(70)
      USAGE IS DISPLAY
.
ADD
RECORD NAME IS IREMARK
RECORD ID IS 0621
LOCATION MODE IS VIA INVOICE-ITEM SET
CALL W02A0001 BEFORE STORE
CALL W02A0001 BEFORE MODIFY
WITHIN AREA CUST-AREA OFFSET 0 PERCENT FOR 100 PERCENT
.
02 TEXT-REMARK
      PICTURE IS X(100)
      USAGE IS DISPLAY
.
ADD
SET NAME IS IX-CUST-NAME
ORDER IS SORTED
MODE IS INDEX USING SYM-CUST-NAME
OWNER IS SYSTEM
      WITHIN AREA CUST-AREA OFFSET 0 PERCENT FOR 100 PERCENT
MEMBER IS CUSTOMER
INDEX DBKEY POSITION IS 5
OPTIONAL MANUAL
KEY IS (
      CUST-NAME ASCENDING )
DUPLICATES ARE NOT ALLOWED
UNCOMPRESSED
.
ADD
SET NAME IS CUST-INVOICE
ORDER IS SORTED
MODE IS CHAIN LINKED TO PRIOR
OWNER IS CUSTOMER
NEXT DBKEY POSITION IS 1
PRIOR DBKEY POSITION IS 2
MEMBER IS INVOICE
NEXT DBKEY POSITION IS 1
PRIOR DBKEY POSITION IS 2
LINKED TO OWNER
OWNER DBKEY POSITION IS 3
MANDATORY AUTOMATIC
KEY IS (
      INVOICE-NUMBER ASCENDING )
      DUPLICATES ARE NOT ALLOWED
      NATURAL SEQUENCE
.
```

```
ADD
SET NAME IS INVOICE-ITEM
ORDER IS SORTED
MODE IS CHAIN LINKED TO PRIOR
OWNER IS INVOICE
NEXT DBKEY POSITION IS 1
PRIOR DBKEY POSITION IS 2
MEMBER IS ITEM
NEXT DBKEY POSITION IS 1
PRIOR DBKEY POSITION IS 2
LINKED TO OWNER
OWNER DBKEY POSITION IS 3
OPTIONAL AUTOMATIC
KEY IS (
      PRODUCT-NUMBER ASCENDING )
      DUPLICATES ARE NOT ALLOWED
      NATURAL SEQUENCE
.
ADD
SET NAME IS INVOICE-IREMARK
ORDER IS SORTED
MODE IS CHAIN LINKED TO PRIOR
OWNER IS INVOICE
NEXT DBKEY POSITION IS 1
PRIOR DBKEY POSITION IS 2
MEMBER IS IREMARK
NEXT DBKEY POSITION IS 1
PRIOR DBKEY POSITION IS 2
LINKED TO OWNER
OWNER DBKEY POSITION IS 3
MANDATORY AUTOMATIC
KEY IS (
      TEXT-REMARK ASCENDING )
      DUPLICATES ARE FIRST
      NATURAL SEQUENCE
.
```

The resulting MDL commands will be:

```
ADD FILE IDMSCUST TYPE IDMS SCHEMA CUSTSCHM VERSION 1 DBNAME 'CUSTSS01'
ADD RECORD CUSTOMER OF IDMSCUST SIZE 45 STORAGE-KEY CUST-NUMBER STORAGE-AREA CUST-
AREA
ADD FIELD CUST-NUMBER OF CUSTOMER POSITION 1 SIZE 10 TYPE CHARACTER
ADD FIELD CUST-NAME OF CUSTOMER POSITION 11 SIZE 20 TYPE CHARACTER
ADD FIELD CUST-CITY OF CUSTOMER POSITION 31 SIZE 15 TYPE CHARACTER
ADD RECORD INVOICE OF IDMSCUST SIZE 20 STORAGE-KEY INVOICE-NUMBER STORAGE-AREA
CUST-AREA
ADD FIELD INVOICE-NUMBER OF INVOICE POSITION 1 SIZE 10 TYPE CHARACTER
ADD FIELD INVOICE-DATE OF INVOICE POSITION 11 SIZE 10 TYPE CHARACTER
ADD RECORD ITEM OF IDMSCUST SIZE 75 STORAGE-AREA CUST-AREA
ADD FIELD PRODUCT-NUMBER OF ITEM POSITION 1 SIZE 5 TYPE CHARACTER
ADD FIELD PRODUCT-DESCRIPTION OF ITEM POSITION 6 SIZE 70 TYPE CHARACTER
ADD RECORD IREMARK OF IDMSCUST SIZE 100 STORAGE-AREA CUST-AREA
ADD FIELD TEXT-REMARK OF IREMARK POSITION 1 SIZE 100 TYPE CHARACTER
ADD INDEX IX-CUST-NAME BASED ON CUST-NAME
ADD LINK CUST-INVOICE FROM CUSTOMER TO (INVOICE)
ADD LINK INVOICE-ITEM OPTIONAL FROM INVOICE TO (ITEM)
ADD LINK INVOICE-IREMARK FROM INVOICE TO (IREMARK)
```

A.3. IDMS RECORD

When starting the Collect File functionality of the MetaStore Manager the option "IDMS Record punch" represents a possibility to transform IDD record definitions into a MetaSuite definition.

Source information

You will first need to produce a Record punch from the CA-IDMS IDD. This can be done by the following example JCL:

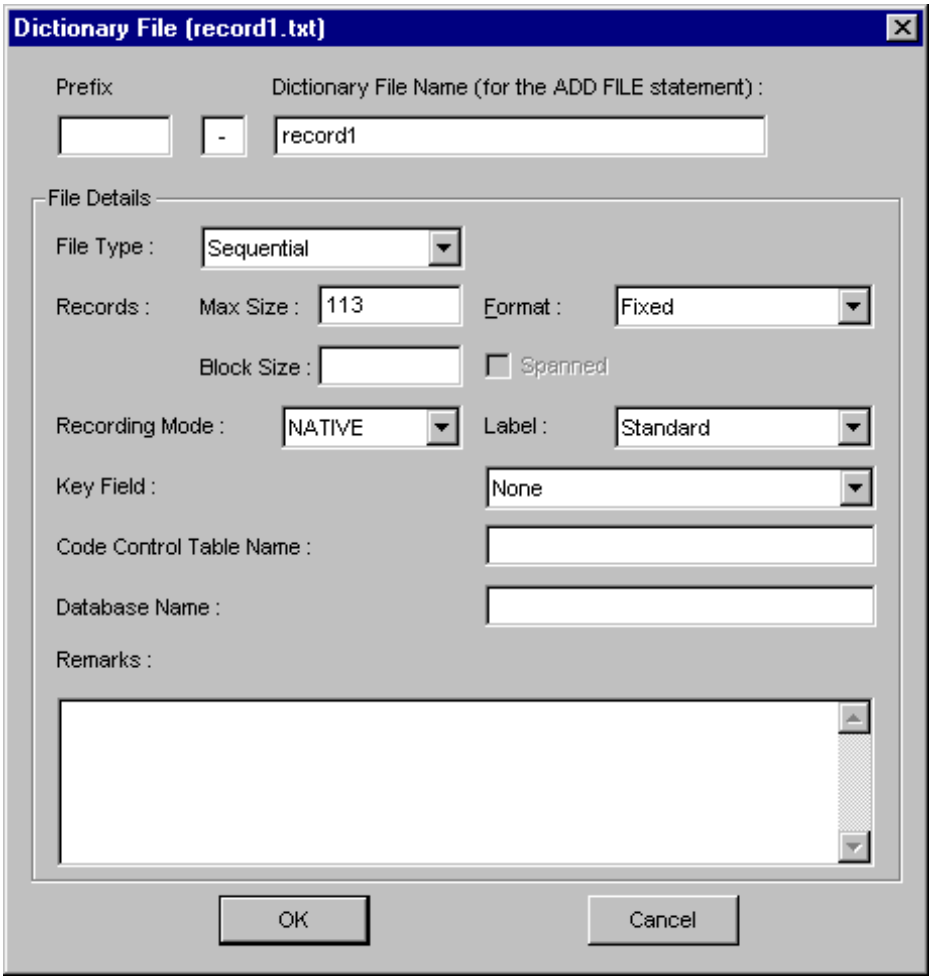
The input is an IDMS punch of your record as a result of the following example JCL for z/OS:

```
//RECCHEM EXEC PGM=RECSCHEM
//SYSCTL DD DSN=IDMS-SYSCTL-filename,DISP=SHR
//SYSUDUMP DD SYSOUT=R
//SYSLST DD SYSOUT=R
//SYSPCH DD SYSOUT=R
//SYSIPT DD *
SIGNON … DICTNAME IS IDMS-DatabaseName
SET OPTIONS FOR SESSION INPUT COLUMNS ARE 1 THRU 80.
   PUNCH RECORD RecordName VERSION IS VersionNumber
    WITH COBOL
    ALSO WITH DETAILS
    ALSO WITH SYNONYMS
    AS SYNTAX.
```

The result of this JCL will need to be transferred to your workstation, so that it is available as source for MetaStore Manager.

IDMS File Information

After starting the "IDMS Record punch" collect option and selecting the IDMS Record punch, the following window will pop up.



Result

A MetaSuite file will be produced which represents the correct file definition as stored in IDD.

Example

Consider the following IDMS Record punch

```
ADD
     RECORD NAME IS KX0001RS VERSION IS 1
        DESCRIPTION IS 'Control and transformation of
        date'
*+      RECORD LENGTH IS 113
        RECORD NAME SYNONYM IS KX0001RS VERSION 1
            SUFFIX IS -KX0001RS
        COMMENTS
```

```
            ' '
      - 'Only returncode zero is ok'
         .
      02 INTERFACEELE
         USAGE IS DISPLAY
*+       ELEMENT LENGTH IS 93
*+       POSITION IS 1
         .
         03 RECID
            PICTURE IS X(8)
            USAGE IS DISPLAY
*+          ELEMENT LENGTH IS 8
*+          POSITION IS 1
            .
         03 RETURNCODE
            PICTURE IS X
            USAGE IS DISPLAY
*+          ELEMENT LENGTH IS 1
*+          POSITION IS 9
            .
         03 MELDING
            PICTURE IS X(80)
            USAGE IS DISPLAY
*+          ELEMENT LENGTH IS 80
*+          POSITION IS 10
            .
         03 STUCOD
            PICTURE IS X(04)
            USAGE IS DISPLAY
*+          ELEMENT LENGTH IS 4
*+          POSITION IS 90
            .
      02 DATUM
         PICTURE IS X(10)
         USAGE IS DISPLAY
*+       ELEMENT LENGTH IS 10
*+       POSITION IS 94
         .
      02 DATUMTYP
         PICTURE IS X
         USAGE IS DISPLAY
*+       ELEMENT LENGTH IS 1
*+       POSITION IS 104
         .
      02 INDKEY
         PICTURE IS X
         USAGE IS DISPLAY
*+       ELEMENT LENGTH IS 1
*+       POSITION IS 105
         .
      02 DATUMUIT
         PICTURE IS 9(8)
         USAGE IS DISPLAY
*+       ELEMENT LENGTH IS 8
*+       POSITION IS 106
         .
```

Appendix B - Sample Programs

B.1. MDL data Samples

MDL definition of IDMS-SIMULATION-CARS

```
DELETE FILE ALL
ADD FILE IDMS-SIMULATION-CARS TYPE IDMS SCHEMA CARSCHEM VERSION 1 DBNAME 'CARS'
ADD RECORD CAR-TYPES#30288 SIZE 24 STORAGE-AREA CARS
ADD FIELD CAR-TYPE-CODE#30290 POSITION 1 SIZE 4 TYPE ZONED UNSIGNED CODE
ADD FIELD CAR-TYPE-NAME#30291 POSITION 5 SIZE 20 TYPE CHARACTER
ADD RECORD CAR-SUBTYPES#30289 SIZE 34 STORAGE-AREA CARS
ADD FIELD CAR-SUBTYPE-MAIN-CODE#30292 POSITION 1 SIZE 4 TYPE ZONED UNSIGNED CODE
ADD FIELD CAR-SUBTYPE-SUB-CODE#30293 POSITION 5 SIZE 6 TYPE CHARACTER
ADD FIELD CAR-SUBTYPE-BUILD-YEAR#30310 POSITION 11 SIZE 4 TYPE ZONED UNSIGNED
ADD FIELD CAR-SUBTYPE-NAME#30294 POSITION 15 SIZE 20 TYPE CHARACTER
ADD INDEX CAR-IX BASED ON CAR-TYPE-CODE#30290
ADD LINK CAR-REL FROM CAR-TYPES#30288 TO (CAR-SUBTYPES#30289)
```

MDL definition of IDMS-SIMULATION-GARAGES

```
ADD FILE IDMS-SIMULATION-GARAGES TYPE IDMS SCHEMA GARSCH VERSION 1 DBNAME 'GARAGES'
ADD RECORD GARAGE-AREAS#30301 SIZE 24 STORAGE-KEY GARAGE-POSTAL-CODE#30303 STOR-
AGE-AREA GARAGES
ADD FIELD GARAGE-POSTAL-CODE#30303 POSITION 1 SIZE 4 TYPE ZONED UNSIGNED CODE
ADD FIELD GAR-COMUNITY#30304 POSITION 5 SIZE 20 TYPE ALPHABETIC
ADD RECORD GARAGE-NAMES#30302 SIZE 32 STORAGE-KEY GARAGE-POSTAL-CODE#30306 STOR-
AGE-AREA GARAGES
ADD FIELD GARAGE-POSTAL-CODE#30306 POSITION 1 SIZE 4 TYPE ZONED UNSIGNED
ADD FIELD GARAGE-CODE#30307 POSITION 5 SIZE 4 TYPE CHARACTER
ADD FIELD GARAGE-CAR-TYPE-CODE#30308 POSITION 9 SIZE 4 TYPE ZONED UNSIGNED CODE
ADD FIELD GARAGE-NAME#30309 POSITION 13 SIZE 20 TYPE CHARACTER
ADD INDEX GAR-IX BASED ON GARAGE-POSTAL-CODE#30303
ADD LINK GAR-REL FROM GARAGE-AREAS#30301 TO (GARAGE-NAMES#30302)
```

B.2. Sample 1 - "Via Index"

```
FIELD W-CUR SIZE 4 TYPE BINARY
FIELD T01-TOVI TYPE CHARACTER SIZE 56
FIELD WK-IDMS-SIMULATION-GARAGES TYPE CHARACTER SIZE 17 INITIAL 'GARAGES' PARAME-
TER
FIELD WK-IDMS-SIMULATION-CARS TYPE CHARACTER SIZE 17 INITIAL 'CARS' PARAMETER
REMARKS Dummy WorkField for TOTAL without ACCUM Fields
FIELD SYS-DUMMY TYPE ZONED SIZE 1
REMARKS Source: IDMS-SIMULATION-GARAGES
SOURCEFILE IDMS-SIMULATION-GARAGES -
SCHEMA GARSCH VERSION 1 DBNAME WK-IDMS-SIMULATION-GARAGES -
 PATH -
 ( -
  GARAGE-NAMES#30302 -
 )
REMARKS Source: IDMS-SIMULATION-CARS
SOURCEFILE IDMS-SIMULATION-CARS -
SCHEMA CARSCHEM VERSION 1 DBNAME WK-IDMS-SIMULATION-CARS -
 CONTROLLED MANUAL
REMARKS REPORT 0 to write values of target groupby fields and program procedures

REPORT 0
REMARKS IDMS-TEST-OBTAIN-VIA-IX (SEQUENTIAL)

TARGETFILE 1 SEQUENTIAL
TITLE 0 ('T01-IDMS-TEST-OBTAIN-VIA-IX')
DETAIL 1 RECORD 'TOVI' -
 ( -
  GARAGE-POSTAL-CODE#30306, -
  GARAGE-CODE#30307, -
  GARAGE-NAME#30309, -
  GARAGE-POSTAL-CODE#30306, -
  CAR-TYPE-CODE#30290, -
  CAR-TYPE-NAME#30291 -
 )

BEGIN SOURCEFILE IDMS-SIMULATION-GARAGES INPUT
REMARKS FI1


BEGIN SOURCEFILE IDMS-SIMULATION-CARS INPUT
REMARKS FI2
CAR-SUBTYPE-MAIN-CODE#30292 SYS-RAW = GARAGE-CAR-TYPE-CODE#30308 SYS-RAW
IDMS OBTAIN FIRST CAR-SUBTYPES#30289 WITHIN CAR-REL
IDMS IF CAR-REL MEMBER
IF IDMS-SIMULATION-CARS SYS-IO-STATUS EQ SYS-NOT-RELATED -
  EXCLUDE
IDMS ACCEPT W-CUR FROM CAR-SUBTYPES#30289 CURRENCY
DEBUG 'current car-subtypes key is # ' ( W-CUR )
IDMS OBTAIN OWNER WITHIN CAR-REL
IDMS IF CAR-REL MEMBER
IF IDMS-SIMULATION-CARS SYS-IO-STATUS EQ SYS-NOT-RELATED -
  DEBUG 'record excluded ' -
  EXCLUDE
IDMS ACCEPT W-CUR FROM CAR-TYPES#30288 CURRENCY
DEBUG 'current car-types key is # ' ( W-CUR )
BEGIN REPORT 0 INITIAL
```

```
SYS-APPLICATION = 'p3127a'
BEGIN REPORT 0 EOJ
Sample 2. "Controlled-by (manual)"
REMARKS SPECIAL IDMS-DBMS WORKFIELD(S)
FIELD WK-IDMS-SIMULATION-GARAGES TYPE CHARACTER SIZE 17 INITIAL 'GARAGES' PARAME-
TER
FIELD WK-IDMS-SIMULATION-CARS TYPE CHARACTER SIZE 17 INITIAL 'CARS' PARAMETER
REMARKS Dummy WorkField for TOTAL without ACCUM Fields
FIELD SYS-DUMMY TYPE ZONED SIZE 1
REMARKS Source: IDMS-SIMULATION-GARAGES
SOURCEFILE IDMS-SIMULATION-GARAGES -
SCHEMA GARSCH VERSION 1 DBNAME WK-IDMS-SIMULATION-GARAGES -
 PATH -
  ( -
   GARAGE-AREAS#30301, -
   GARAGE-NAMES#30302 -
   VIA GAR-REL -
  )
REMARKS Source: IDMS-SIMULATION-CARS
SOURCEFILE IDMS-SIMULATION-CARS -
SCHEMA CARSCHEM VERSION 1 DBNAME WK-IDMS-SIMULATION-CARS -
 CONTROLLED BY IDMS-SIMULATION-GARAGES -
 KEY GARAGE-CAR-TYPE-CODE#30308 -
 PATH -
  ( -
   CAR-TYPES#30288 -
   VIA CAR-IX -
  )
REMARKS REPORT 0 to write values of target groupby fields and program procedures

REPORT 0
REMARKS Car-Garage-report (REPORT)

REPORT 1 -
PAGE ( 55, 132 )
TITLE 0 ('Garages and cars')
DETAIL 1 -
( -
 GARAGE-POSTAL-CODE#30303, -
 GAR-COMUNITY#30304, -
 GARAGE-POSTAL-CODE#30306, -
 GARAGE-CODE#30307, -
 GARAGE-CAR-TYPE-CODE#30308, -
 GARAGE-NAME#30309, -
 CAR-TYPE-CODE#30290, -
 CAR-TYPE-NAME#30291 -
)
BEGIN REPORT 0 INITIAL
SYS-APPLICATION = 'P3134A'
BEGIN REPORT 0 EOJ
```

B.3. Sample 2 - "OBTAIN"

```
FIELD T01-TO-rec TYPE CHARACTER SIZE 52
FIELD WK-IDMS-SIMULATION-GARAGES TYPE CHARACTER SIZE 17 INITIAL 'GARAGES' PARAME-
TER
FIELD SYS-DUMMY TYPE ZONED SIZE 1
REMARKS Source: IDMS-SIMULATION-GARAGES
SOURCEFILE IDMS-SIMULATION-GARAGES -
SCHEMA GARSCH VERSION 1 DBNAME WK-IDMS-SIMULATION-GARAGES -
 CONTROLLED MANUAL
REMARKS No automatic file specified
SOURCEFILE SYS-DUMMY-FILE

REMARKS REPORT 0 to write values of target groupby fields and program procedures
REPORT 0
REMARKS test-obtain (HTML)
TARGETFILE 1 DELIMITED OUTPUT-CONTROL HTM
TITLE 0 ('T01-test-obtain')
DETAIL 1 RECORD 'TO-rec' -
( -
 GARAGE-CODE#30307, -
 GARAGE-CAR-TYPE-CODE#30308, -
 GARAGE-NAME#30309, -
 GARAGE-POSTAL-CODE#30303, -
 GAR-COMUNITY#30304 -
)
BEGIN SOURCEFILE IDMS-SIMULATION-GARAGES INPUT
GARAGE-POSTAL-CODE#30306 = 3118
IDMS OBTAIN GARAGE-NAMES#30302 WITHIN GAR-REL USING GARAGE-POSTAL-CODE#30306
IF IDMS-SIMULATION-GARAGES SYS-IO-STATUS EQ SYS-ERROR -
  DEBUG 'OBTAIN FIRST NOT SUCCESSFULL' -
  HALT ALL -
  EXIT
IDMS OBTAIN OWNER WITHIN GAR-REL
IF IDMS-SIMULATION-GARAGES SYS-IO-STATUS EQ SYS-ERROR -
OR IDMS-SIMULATION-GARAGES SYS-IO-STATUS EQ SYS-NOT-RELATED -
  DEBUG 'OBTAIN OWNER NOT SUCCESSFULL' -
  HALT ALL -
  EXIT
BEGIN REPORT 0 INITIAL
SYS-APPLICATION = 'P3146A'
BEGIN REPORT 0 EOJ
BEGIN TARGETFILE 1 INPUT
IF IDMS-SIMULATION-GARAGES SYS-INPUT-COUNT GT 2 -
  HALT ALL
```

B.4. Sample 3 - "Controlled by work field"

```
REMARKS Global WorkFields without LIKE
FIELD WORK-CAR-TYPE-CODE SIZE 4 TYPE ZONED UNSIGNED
REMARKS Global WorkFields for TargetFields
FIELD T01-Cra-Garage-rec TYPE CHARACTER SIZE 114
REMARKS SPECIAL IDMS-DBMS WORKFIELD(S)
FIELD WK-IDMS-SIMULATION-GARAGES TYPE CHARACTER SIZE 17 INITIAL 'GARAGES' PARAME-
TER
FIELD WK-IDMS-SIMULATION-CARS TYPE CHARACTER SIZE 17 INITIAL 'CARS' PARAMETER
REMARKS Dummy WorkField for TOTAL without ACCUM Fields
FIELD SYS-DUMMY TYPE ZONED SIZE 1
REMARKS Source: IDMS-SIMULATION-GARAGES
SOURCEFILE IDMS-SIMULATION-GARAGES -
SCHEMA GARSCH VERSION 1 DBNAME WK-IDMS-SIMULATION-GARAGES -
 PATH -
 ( -
  GARAGE-AREAS#30301, -
  GARAGE-NAMES#30302 -
  VIA GAR-REL -
 )
REMARKS Source: IDMS-SIMULATION-CARS
SOURCEFILE IDMS-SIMULATION-CARS -
SCHEMA CARSCHEM VERSION 1 DBNAME WK-IDMS-SIMULATION-CARS -
 CONTROLLED BY IDMS-SIMULATION-GARAGES -
 KEY WORK-CAR-TYPE-CODE -
 PATH -
 ( -
  CAR-TYPES#30288 -
  VIA CAR-IX, -
  CAR-SUBTYPES#30289 -
  VIA CAR-REL -
 )
REMARKS REPORT 0 to write values of target groupby fields and program procedures

REPORT 0
REMARKS Car-Garage-report (REPORT)

REPORT 1 -
PAGE ( 55, 132 )
TITLE 0 ('Garages and cars')
DETAIL 1 -
( -
 GARAGE-POSTAL-CODE#30303, -
 GAR-COMUNITY#30304, -
 GARAGE-POSTAL-CODE#30306, -
 GARAGE-CODE#30307, -
 GARAGE-CAR-TYPE-CODE#30308, -
 GARAGE-NAME#30309, -
 CAR-TYPE-CODE#30290, -
 CAR-TYPE-NAME#30291, -
 CAR-SUBTYPE-MAIN-CODE#30292, -
 CAR-SUBTYPE-SUB-CODE#30293, -
 CAR-SUBTYPE-BUILD-YEAR#30310, -
 CAR-SUBTYPE-NAME#30294 -
)
BEGIN SOURCEFILE IDMS-SIMULATION-GARAGES INPUT
IF GARAGE-NAMES#30302 SYS-PATH-COUNT EQ 0 -
```

```
  WORK-CAR-TYPE-CODE SYS-STATUS = SYS-NULL-VALUE -
ELSE -
  WORK-CAR-TYPE-CODE = GARAGE-CAR-TYPE-CODE#30308

BEGIN REPORT 0 INITIAL
SYS-APPLICATION = 'P3161A'
BEGIN REPORT 0 EOJ
```