ADABAS File Access Guide

Release 8.1.3

November 2013

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

Table of Contents

CHAPTER 1
About This Manual

MetaSuite file access for ADABAS/C is intended for users with some experience with MetaSuite. The supplement describes the use of the MetaSuite Command Language to access ADABAS/C databases, and Job Control Language (JCL) changes needed to run MetaSuite application programs in the ADABAS/C environment.

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

1.1. Prerequisites

Readers are expected to be familiar with ADABAS/C.

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.

In addition, you may want to refer to the related ADABAS/C manuals.

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

General Database Concepts & Terminology

The concepts and terminology described in this chapter apply to all database management systems. This chapter describes how the MetaSuite generalized database interface views a database, regardless of the particular database management system (DBMS) you are using.

Topics covered:

2.1. Databases and Their Uses

In order to retrieve information about transactions, businesses keep records of those transactions. As long as the amount of data remains manageable, it is easy to gather detailed information about any transaction and to maintain a global view of all transactions. However, increasing amounts of data strain the ability of any system to process the information to suit all purposes. It becomes more efficient in terms of the process to group information by application. Computer files are usually organized by application, with application programs designed specifically to process those files. The programs depend upon the physical organization of the files. Many files contain redundant data, with each copy of the data requiring effort to be updated and maintained. In time, because of the number and diversity of application programs, the data files and retrieval systems can no longer respond to requests for information and the need for change in a timely fashion.

At this point, a different outlook on the problem of data storage and retrieval is required. This is where database management systems (DBMSs) come in. A database management system attempts to co-ordinate the data requirements for several departments and applications in order to make data more accessible, control data redundancy and allow the database to be independent of the application programs which use it. This facilitates the maintenance of the data itself, because data can be added, modified and retrieved in a uniform fashion. In addition, a DBMS handles the internal processing of data relationships and retrieval, which makes the system more flexible and responsive to ad hoc requests for information. Most database systems also provide certain privacy and security functions for the protection of data.

2.2. Basic Database Concepts

A database consists of data and relationships, which are collectively known in this document as entities. Each database management system uses some combination of these two primitive entities to provide four or five higher-level entities. MetaSuite users deal with the five higher-level entities, and MetaSuite translates the user-specifications into the correct format for the particular DBMS.

The five database entities that MetaSuite provides are:

- files
- records
- fields
- links
- indexes

Each DBMS will treat each of these entities as data or as relationships in processing user requests, and each DBMS categorizes them differently. The MetaSuite user needs not be concerned with the actual DBMS categorizations -- the usage of each of these entities in the MetaSuite language is the same, regardless of the type of DBMS.

When speaking of entities, we may refer to any of three concepts: the entity type, the entity occurrence or the entity value. The discussion that follows first defines the "type", "occurrence" and "value" concepts, and then goes onto describe the entity types provided by the MetaSuite database interface.

Entity Type

The entity type is identified by a name: CUSTOMER *record*, CUST-NUMBER *field*, etc. Another word for "type" is "definition". As an analogy, consider the universe of containers. There are boxes, cartons, jars, buckets, and many more. The name "shoe box", for instance, refers to a particular type of container, which always has a specific shape, size, and function. Similarly, in the universe of database entities, each has a name that refers to a specific definition and function.

Field

A field is the basic data entity. Each occurrence contains a value (or series of values). The simplest kind of field is the elementary field, which contains a single value at any one time. Next is the group field, which contains several elementary fields, known as subfields. Every occurrence of the group field will have the same collection of subfield occurrences, in the same sequence. An occurring field is like a simple field, except that it holds a series of values. There may be occurring groups and group subfields. The smaller containers in our analogy are the fields. A box containing one item would be analogous to an elementary field, a box of assorted jars would be like a group field, and a box of identical jars would be like an occurring field.

Record

A record is the next kind of entity. A record is a collection of fields. Each occurrence of a record type consists of a set of field occurrences, in a fixed sequence. The types and sequence of the fields will be the same in every occurrence of the record type. In MetaSuite, users always think of records as data entities -- thus in the analogy, the corresponding kind of entity would be a shipping carton. It does not matter what scheme the DBMS uses to store the record entity -- MetaSuite automatically translates the user's record references into the proper format for the DBMS in use.

Note: Every database contains an arbitrary number of occurrences of a fixed number of record types. This (to choose another analogy) is like a bakery: the bakery items are like database records. There are a fixed number of types of items in the bakery, but a large number of occurrences of each type.

Link

A link is an entity that relates records to one another. The link relates occurrences of two record types to one another. The link may be one-to-many, in which case one occurrence of one record type is related to several occurrences of another record type. An analogy to this is the relationship between mother and child. Each child has a single (biological) mother, and each mother may have one or several children. The mothers and children are like the records of a database, and the "mother/child" relationship between them is a type of one-to-many link relationship. The other kind of link is the many-to-many link, in which case several occurrences of a record type are each related to several occurrences of another record type. An analogy to this is the relationship between uncles and nieces. Each uncle may have one or more nieces, and each niece may have one or more uncles.

Index

An index is an entity that also relates records to one another. It is different from a link, in that a link relates small groups of occurrences of different record types, while an index relates all of the occurrences of a single record type. Thus, there is only one occurrence of each index type. Furthermore, there is often no specific order to the record occurrences related in a link, whereas the record occurrences related in an index are always in sorted sequence. To choose our last analogy, consider a card catalog in a library. In the library, every book is described by three index cards. The index cards are separated into three card catalogs, each of which is sorted in a different sequence (subject, author, and title). The books in the library are like the records in the database, and the three card catalogs are like the indexes.

File

Finally, a file is a collection of records, fields, links, and indexes. In non-database systems, this collection of entities is stored in a single dataset by the operating system. In some database systems, a file similarly consists of the entire collection of all those entities, while in other database systems, the collection may be limited by the DBMS to a certain "view" of the database which contains only certain records, fields, links and indexes.

Entity Occurrence

The second important database concept is that of occurrence. An entity occurrence is a particular instance of some entity type. In the database, it is a specific CUSTOMER record or a specific CUST-NUMBER field. In our "container" analogy, the occurrence refers to a particular container. Each DBMS uses a different system for identifying the entity occurrences -- some use no system at all. It is not important for you, the user, to be able to identify a unique entity occurrence to the DBMS. What is important is to understand the difference between the concepts of "type" and "occurrence", as these will be used throughout this supplement.

Entity Value

The third and last important database concept is that of value. In a database, the value of a data entity is what is stored in a particular occurrence of the entity type (e.g., CUST-NAME occurrence 110472 might contain the value 'JOHN Q. DOE'). In our analogy, the value of a container is simply the contents of the container.

2.3. Types of Database Management Systems

Each DBMS system provides some set of functions which may be performed on the occurrences and values of the entities stored in the database. Every function has three characteristics: the operation, the entity types to be processed, and the entity occurrences to be processed. The typical operations are retrieval, storage, modification and deletion. The way in which MetaSuite identifies to the DBMS the types and occurrences to be processed places the DBMS into one of two categories: navigational or relational. Each of these is described below.

Navigational DBMSs

In navigational DBMSs, processing is performed in increments, one entity occurrence at a time. The occurrences are identified relative to the current position in the database (e.g. "read the next record"). Schematic diagrams of the databases processed by this kind of DBMS resemble road maps. In order to perform a complex function, it is necessary to "navigate" this "road map", hence the name "navigational".

Relational DBMSs

In relational DBMSs, all of the processing is performed at once -- many entity occurrences at a time. The occurrences are identified by data entity values and relationship entity occurrences (e.g., "obtain all of the orders related to all customers with zipcode 01455"). The identification operators are those of relational calculus (AND, OR, NOT, UNION, INTERSECT, JOIN), hence the name "relational".

2.4. The MetaSuite Generalized Database Interface

As with all other file types, the process of using databases with MetaSuite consists of two steps:

- Step 1: Defining the data
- Step 2: Accessing the data

A database file should be defined to the MetaSuite Dictionary by someone familiar with the concepts and facilities of the particular DBMS system in use. However, MetaSuite users who will only write programs to access the database need only understand the fundamental concepts of the MetaSuite database interface: the file, record, field, link and index entities. They do not need to be familiar with the concepts and facilities of any particular DBMS, because the MetaSuite program definition language is DBMS-type independent. These concepts, in one form or another, are common to all database systems (with one or two exceptions), and are automatically converted by the MetaSuite system into the proper DBMS functions (either navigational or relational expressions).

CHAPTER 3

The ADABAS/C Database System

ADABAS/C is the database management system (DBMS) distributed by Software AG of North America. It supports the MetaSuite database entity concepts of field, record, link, and index.

There is no concept in ADABAS/C that corresponds to the MetaSuite file concept, so the entire ADABAS/C database is treated as a single MetaSuite file.

Topics covered:

3.1. ADABAS/C Databases

An ADABAS/C database consists of one or more ADABAS/C files, and is defined to the MetaSuite Dictionary as one or more files of the type "ADABAS". Throughout this document, a MetaSuite file which defines an ADABAS/C database will be referred to as a "MetaSuite (ADABAS/C) file" to distinguish it from a single ADABAS/C file.

The ADABAS/C files that make up an ADABAS/C database are typically related to one another. The relationships are recorded in the database by storing common identifier values in the record occurrences of different ADABAS/C files. For instance, a CUSTOMER file and an ORDER file might be related by storing a customer account number in every record of both files -- those customer and order records with matching customer account numbers are related. If the ADABAS/C DBMS is informed of this relationship, then the ADABAS/C files are said to be "coupled".

Whether or not the ADABAS/C files are "coupled", all relationships between the files of an ADABAS/C database should be defined to the MetaSuite Dictionary as link definitions, so that MetaSuite users can make full use of those relationships.

3.2. ADABAS/C Files

The data stored in ADABAS/C databases is stored in simple files called ADABAS/C files. Generally, each ADABAS/C file contains all the occurrences of a particular record type. A typical database will contain several ADABAS/C files. In simple databases, each ADABAS/C file will correspond to a MetaSuite record. In such databases, all of the file types that appear together in an ADABAS/C file form a single record type.

Sometimes, however, a single ADABAS/C file may be used to contain two or more record types. This type of ADABAS/C file is called a multi-record file in the ADABAS/C literature, and is maintained through the use of the ADABAS/C "null value suppression" feature. The DBMS considers all of the fields in a multi-record ADABAS/C file to be a single record type, part of which may be missing in any one occurrence.

Note: It is important to note that the individual record types are not defined separately to the ADABAS/C DBMS, but rather are maintained by the application programs that update the database. In order to provide the full power and flexibility of MetaSuite, however, each of the individual record types should be defined to MetaSuite as a separate record. In that case, a single ADABAS/C file will correspond to several MetaSuite records.

3.3. Record Identifier Keys

In order for MetaSuite to distinguish the different records in an ADABAS/C multi-record file, it needs the description of record identifier keys. These are defined to MetaSuite with the KEY clause of the ADD RECORD command, just as for non-database files. MetaSuite will use the record identifier keyfield to select the proper record types when processing ADABAS/C files.

3.4. Fields

All of the individual data elements which are stored in an ADABAS/C file are defined to the DBMS as fields. Any data element defined to ADABAS/C may be defined to MetaSuite as a field. Sometimes a data element will be defined to ADABAS/C as a simple field, when in fact it consists of several subfields.

These subfields may also be defined to MetaSuite as fields. MetaSuite will automatically formulate the proper ADABAS/C request and extract the desired data elements, regardless of whether or not the field is actually defined to ADABAS/C. In other DBMS environments, there is a specific "record layout" which is defined to the DBMS, and which applications use in accessing the data. There is no such concept in ADABAS/C – the applications are free to specify their own record layouts on every request. MetaSuite users will define the entire database record to the MetaSuite Dictionary, and MetaSuite will automatically construct the proper request and record layout on a program by program basis.

The ADABAS/C field may be defined as:

- an individual data element,
- a group of elements,
- a "periodic group" of elements,
- a "multi-valued field".

Individual elements and groups correspond exactly to the MetaSuite concept of fields and groups.
A **periodic group** is defined to MetaSuite as a group field with an OCCURS clause.
A **multi-valued field** is defined to MetaSuite as a simple field with an OCCURS clause, where the number of occurrences specified in the MetaSuite ADD FIELD command is equal to the maximum number of values stored in the multi-valued field.

3.5. Associator

ADABAS/C maintains an extensive set of indexes for each of the ADABAS/C files in a special file called the associator. This is where the DBMS maintains the associations between the user's data. The associator may be thought of as a collection of indexes (just like the index portion of a VSAM cluster). The associator holds many of these indexes, which in ADABAS/C terminology are known as descriptors.

3.6. ADABAS/C Descriptors

Each descriptor is simply a list of pairs of the form "value,ISN", kept in sorted sequence. In a simple descriptor, each value is a copy of the value of one field taken from one record occurrence in one of the ADABAS/C files. The "ISN" is a number used by ADABAS/C to identify the particular record occurrence. The list is kept in sorted sequence, where the data value is the sort key. There are other types of descriptors, and all of them have the same form: they are a list of "value,ISN" pairs.

The other types of descriptors are known as superdescriptors, subdescriptors, and phonetic descriptors. The value portion of each superdescriptor entry is formed from the data in several fields in each record occurrence. Each value in a subdescriptor is a portion of a field in each record occurrence. Each value in a phonetic descriptor is a translation of an alphabetic field in each record occurrence.

When multi-valued fields are used as descriptors, ADABAS/C treats them like their name: a single field (hence a single descriptor) with multiple values. An ORDER record containing a multi-valued descriptor field called ITEM-TYPE whose values are, for instance, HAMMER, PAIL, and BRUSH, could be accessed through the ITEM-TYPE descriptor. The record would be retrieved once among the orders for hammers, again among the other orders for pails, and a third time among the other orders for brushes. This is a very powerful capability, which MetaSuite exploits fully.

Descriptors will be used by MetaSuite to:

- Sequentially read the database in sorted sequence,
- Perform direct access to the database, and
- Retrieve records that are related to one another.

For these purposes, MetaSuite supports the use of simple descriptors, superdescriptors, subdescriptors and phonetic descriptors.

CHAPTER 4
Defining the Dictionary

To construct MetaSuite application programs which access ADABAS/C databases, you will need to do two things:

1. Define the database entities to the MetaSuite Dictionary.
2. Use those entities with the MetaSuite language.

This chapter describes the process of defining the database entities to the MetaSuite Dictionary.

Topics covered:

4.1. Note to VSE/AF Users

MetaSuite application programs normally use some of the same system-names (SYS001, etc.) in the execution JCL as the ADABAS/C system does. You will need to modify the MetaSuite-System to use different system-names. The procedure for doing this is described in the VSE/AF JCL considerations. See Program Generation on page 28.

The modification is quite simple, but will affect the execution JCL needed for all MetaSuite applications -- both those that access the ADABAS/C database and those that do not. You should make the modification when you install MetaSuite, so that all applications will use the modified JCL.

Otherwise, you will have to modify the execution JCL of any existing program that is regenerated after you begin using the MetaSuite ADABAS/C interface.

4.2. Sources of Information

The first step is to locate the sources of information about the database definition:

- ADABAS/C Data Base Status Report (DBSR) produced by the ADABAS/C ADAREP utility.
- ADABAS/C Data Dictionary DD07 report produced by the ADABAS/C DDREPORT utility.
- ADABAS/C Data Definition Module (DDM) source-statements.

ADABAS/C users who have the ADABAS/C Data Dictionary facility will use the DBSR and the DD07, others will use the DBSR and the DDM source.

Using the Data Base Status Report

The DBSR contains a description of all the ADABAS/C files available in the database. It consists of a database section followed by several file sections. The beginning of the database section lists the files available, giving their ADABAS/C file numbers and names. You will need both of these pieces of information for each file.

Each file section of the DBSR contains a Field Descriptor Table. This is the master guide to the ADABAS/C file contents. For simple ADABAS/C files, all of the fields in the file are defined to MetaSuite as a single record, with the fields in the sequence shown in the Field Descriptor Table. For multi-record ADABAS/C files, the fields are grouped into two or more subsets, each of which will be defined to MetaSuite as a separate record. Unfortunately, there is no DBMS documentation which will indicate how to organize the groupings -- this must be provided by the database systems staff at your site.

Each entry in the Field Descriptor Table describes a single field in the ADABAS/C file. The description consists of the following characteristics:

- level
- name
- length
- format
- options

You will need all of these in order to specify the MetaSuite FIELD definition. Note that position is not a characteristic. You should define the first level 1 field in each record as being POSITION 1, and omit position information for all subsequent level 1 fields (MetaSuite will calculate a suitable value).

Position for a subfield should be calculated relative to the group field that contains it (the first position within a group field being POSITION 1).

Using the Data Dictionary Report

If your site has the ADABAS/C Data Dictionary facility, and the database of interest is defined in that data dictionary, then some additional information about the fields may be obtained by running the ADABAS/C DDREPORT utility to produce a DD07 report. The DD07 report lists field definitions, grouped by ADABAS/C file.

Locating the proper file-section, and then matching on name/length/format may find the additional information for a given field. The additional information consists of an external name, the number of decimal places (for numeric fields), an edit-mask (described by a code-value), default headings and comments.

Using the Data Definition Module

ADABAS/C users who do not have the ADABAS/C Data Dictionary facility may still have to obtain additional information about each field. If your systems staff has prepared a DDM for the file of interest (which is used by the ADABAS/C report generators), then the extra information will be in the DDM source statements. The external name, decimal places, edit mask, headings and comments are all provided.

4.3. Defining an ADABAS/C Database to MetaSuite

The ADD FILE command is used to define an ADABAS/C database to the MetaSuite Dictionary. The ADABAS/C DBSR describes the various ADABAS/C files that are available. There is no simple rule for determining which ADABAS/C files to define as a single MetaSuite (ADABAS/C) file. The following steps should be used to determine what you should define as your MetaSuite (ADABAS/C) files:

1. Determine which, if any, of your ADABAS/C files are password-protected. Make a series of (one or more) lists of the various ADABAS/C files at your site, where each list names all of the ADABAS/C files that have the same password - your first (and possibly only) list would be for the ADABAS/C files which are not password protected.
2. For each list of ADABAS/C files consult with your ADABAS/C administrator and determine if any of the files in that list are related to one another. Two ADABAS/C files are related if the value of a field in one file is equivalent to the value of a descriptor on the other file. You may find that the relationships result in "cluster" of related files. For example, consider the following files:

File	Field or Descriptor
CUSTOMER	Customer number descriptor
ORDER	Order number descriptor, Customer number field
LINEITEM	Order number field
PROJECT	Project number descriptor
PERSON	Project number field

   In this case, CUSTOMER, ORDER and LINEITEM are related to one another, and PROJECT and PERSON are related to one another.
3. Each set of ADABAS/C files that form a "cluster" (as defined above) should be defined to the MetaSuite Dictionary as a FILE. The relationships will be defined to MetaSuite as LINKS, and the individual record types will be defined as RECORDS.

After determining how to structure your MetaSuite (ADABAS/C) file definitions, you will use the ADD FILE command to define them to the MetaSuite Dictionary. The syntax for the ADD FILE command when used with non-ADABAS/C files may be found in the *MetaStore Manager User Guide*. The syntax for the ADD FILE command when used to define MetaSuite (ADABAS/C) files is described below.

Note: TYPE, CODE-CONTROL, and REMARKS options are the only options used in this context.

Command

```
ADD FILE file-name
   [VERSION File-version]
   TYPE ADABAS
   [CODE-CONTROL table-name]
   [REMARKS text]
```

Naming the File

file-name

file-name is an arbitrary name that you choose to identify this file to MetaSuite. In the case of a simple database, this might be the database name from the DBSR and DD07 reports.

File-version

[VERSION *File-version*]
Optional.
The *VERSION* option specifies the version number of an ADABAS/C file.

Specifying the Type of File Organization

`TYPE ADABAS`
The TYPE option specifies the file organization of the file being defined. TYPE ADABAS is the only TYPE option allowed for ADABAS/C files.

Indicating the Presence of Alternate Code Generation Tables

`[CODE-CONTROL table-name]`
The generation control (CODE-CONTROL) option allows you to override the normal system generated COBOL code as directed by the dictionary table, *table-name*. This is used when you want to replace or supplement the COBOL code generated by MetaSuite to process the ADABAS/C database. If used, you must create the table specified by table name before generating any programs using this file. This option will typically not be used when processing an ADABAS/C database. For assistance in using this feature, contact MetaSuite Technical Support.

Example

The database name shown in the example DBSR and DD07 reports above is "WIDGET SALES DB". Converting this to a legal MetaSuite name gives the following command:

```
ADD FILE WIDGET-SALES-DB TYPE ADABAS
```

4.4. Defining Records

When you define an ADABAS/C record to the MetaSuite Dictionary, you use the ADD RECORD command. The syntax of the ADD RECORD command when used with non-database records may be found in the *MetaStore Manager User Guide*. The syntax of the ADD RECORD command when used to define database records includes additional options necessary for database processing and is described below.

Command

```
ADD RECORD record-name]
      [OF (file-name, ...)]
      SIZE maximum-record-size
      [KEY (field-name = value)]
      [STORAGE-KEY storage-keyfield]
      DBNAME 'fn'
      [REMARKS text]
```

Naming the Record

record-name

Record-name is an arbitrary name that you choose to identify this record to MetaSuite. In the case of a simple ADABAS/C file, the file name specified in the DBSR database section should be used for the MetaSuite record name; you will define all of the fields that appear in this ADABAS/C file as fields of this MetaSuite record. In the case where the ADABAS/C file is a multi-record file, you will have to consult with your systems staff to determine which fields participate in the various records which are maintained in the ADABAS/C file, and you will define each record as a separate MetaSuite record.

Identifying the File(s) Containing the Record

[OF (*file-name*,...)]

A record may be included in any number of files. The *OF-file-name* option identifies the files containing the named record. If this clause is omitted, the record is assumed to occur only in the preceding file. The name you specify will be the name given the corresponding MetaSuite (ADABAS/C) file.

Specifying the Maximum Record Size

SIZE *maximum-record-size*

The SIZE option specifies the combined size of all of the fields defined for this record. You may estimate this number, but it must be at least as large as the sum of the sizes of all the level-1 fields described in the DBSR report.

Note: The SIZE option is required for records of database files, but is optional for non-database files.

Specifying a Record Identification Field

[KEY (*field-name = value*)]

The KEY clause is used to specify a field and a specific value of that field which identifies this record type. This option is used only for multi-record ADABAS/C files. Each different record in the multi-record file is defined as a separate MetaSuite record, and each requires the KEY clause. Any field named in a KEY clause is assumed by MetaSuite to be a simple descriptor or a subdescriptor field. The record identification field must be defined to the dictionary within this record before generating any programs that access the record. Record identification fields are not identified as such on the DBSR and DD07 reports, so you will have to obtain this information from your systems staff.

Note: The record identification field identifies a type of record, whereas the storage-keyfield (described below) is used to identify a unique occurrence of a specific record type.

Specifying a Storage Keyfield

`[STORAGE-KEY storage-keyfield]`

The STORAGE-KEY option names a field that uniquely identifies a particular occurrence of this record type. Not all record-definitions will have the STORAGE-KEY option, because a field which uniquely identifies a record occurrence will not be present in all record types. Any field named as a *storage-keyfield* is assumed by MetaSuite to be a simple descriptor field. Storage-keyfields are not identified as such on the DBSR and DD07 reports, so you will have to determine this information from your systems staff.

Note: The storage-keyfield is used to identify a unique occurrence of a specific record type, while the record identification field (described above), is used to identify a type of record.

Specifying the Internal ADABAS/C Number

`DBNAME 'fn'`

The required DBNAME option is used to specify the number that identifies the ADABAS/C file in which this record appears. It must be enclosed in quotes, and may be any positive integral number in the range 1 to 255. It must be the same as the file number specified in all the ADABAS/C definition reports.

Examples

In the example database shown earlier, there are three ADABAS/C files: CUSTOMER, ORDER-ITEM, and SALESPERSONS. After consulting with the systems staff at our example site, we discover that ORDER-ITEM is really a multi-record file containing two record types. We will thus define four records to MetaSuite. The record sizes are determined by summing the sizes of all the level-1 fields that appear in each of the records. Note that the record sizes could have been estimated, so long as the estimated sizes were large enough to contain all of the level-1 fields in each record. The DBNAME values are the ADABAS/C file numbers for the ADABAS/C files that contain the records. Our systems staff informs us that the first field (OA) in the ORDER-ITEM file is the record identification (KEY) field, and that value of 'O' and 'I' indicates ORDER and ITEM records, respectively. We will be defining links later in the Dictionary Maintenance, which will require that storage-keyfields be defined for CUSTOMER and SALESPERSON records, and so we will specify storage-keyfields here. Our systems staff has informed us that the fields whose values are used to identify the specific record occurrences of these two record types are CUSTOMER-NUMBER and SALESPERSON-NUMBER, respectively. The four record definition commands are as follows:

```
ADD RECORD ORDER OR WIDGET-SALES-DB SIZE 34 -
   KEY (ORDER-RECTYPE = 'O') DBNAME '01'
ADD RECORD ITEM OF WIDGET-SALES-DB SIZE 30 -
   KEY (ITEM-RECTYPE = 'I') DBNAME '01'
ADD RECORD CUSTOMER OR WIDGET-SALES-DB SIZE 36 -
   STORAGE-KEY CUSTOMER-NUMBER DBNAME '02'
ADD RECORD SALESPERSON OF WIDGET-SALES-DB SIZE 148
   STORAGE-KEY SALESPERSON-NUMBER DBNAME '03'
```

4.5. Defining Fields

The ADD FIELD command is used to define fields within the record. The syntax is identical to that described in the *MetaStore Manager User Guide*, with one exception: every field defined to MetaSuite which is also defined in the DBSR must have a DBNAME option added to the MetaSuite ADD FIELD command. *MetaStore Manager User Guide* for a complete description of all the options of the ADD FIELD command. The discussion of the ADD FIELD command which follows includes only those options for which special considerations apply when defining ADABAS/C fields to the MetaSuite Dictionary.

Abbreviated Command

```
ADD FIELD field-name
      [OF {record-name | group-field}]
      [POSITION start-value]
      SIZE characters
      [OCCURS number-times]
      [DBNAME 'en']
```

Naming the Field

`field-name`

field-name is the name to be assigned to the field in the MetaSuite Dictionary. If your site has the ADABAS/C Data Dictionary, this should be the same as the external name defined for this field in the DD07 report.

Defining a Field Position

`[OF {record-name | group-field}] [POSITION  start-value]`

If the field position options are omitted, the field being defined immediately follows the previously defined field in the current record. The OF *record-name/group-field* ..POSITION option specifies the name of a previously defined MetaSuite record or field in which this field appears. The POSITION option specifies the starting character position of the field relative to the beginning of the record, or the beginning of the named group-field, if coded. When defining ADABAS/C fields, code "OF record-name POSITION 1" for the first field in a record, and omit the field position options for the remainder of the fields except where the "OF group-field" option is required to define a subfield.

Defining the Field's Size

`SIZE characters`

The SIZE option is required for all field definitions, and specifies the length of the field being defined, in terms of characters. The size of a field can be determined from the "length" characteristic shown in the DBSR report.

Defining an Occurring Field

`OCCURS number-times`

The OCCURS option is used to specify the maximum number of occurrences of a repeating field within a record. For ADABAS/C fields, this option must be used to specify the maximum number of occurrences of a periodic group field, or the maximum number of values of a multi-valued field. If your systems staff can provide you with accurate information on the actual maximum in the database, then code that value; otherwise code "OCCURS 99".

Specifying the Internal ADABAS/C Field Name

`[DBNAME 'en']`

The DBNAME option is used to specify the 2-character ADABAS/C field name, which is the name by which the field is identified in all the ADABAS/C definition reports. You must code the DBNAME clause in every MetaSuite field definition that corresponds to a DBSR field. Any subfields that are not defined in the DBSR must not be defined to MetaSuite with a DBNAME.

For instance, if the DBSR and DD07 reports describe a 10-digit field called TELEPHONE-NUMBER, and you want to define MetaSuite subfields for AREA-CODE and PHONE-NUMBER, then TELEPHONE-NUMBER would be defined with a DBNAME, but AREA-CODE and PHONE-NUMBER would not be.

Sources of Field Definition Information

The following table summarizes where the information needed to complete MetaSuite field definitions may be obtained.

Characteristic	Source	Use in MetaSuite
Length	DBSR	Use as SIZE value
Format	DBSR	Converts into TYPE Value
Edit Mask	DD07/DDM	Converts into EDIT Value
Decimals	DD07/DDM	Use as DECIMAL value
Headings	DD07/DDM	Use as HEADING value
Comments	DD07/DDM	Use as REMARKS text

Examples

Looking at our example DBSR and DD07 reports, we now construct the field definition commands for the four sample database records. The commands for the ORDER record fields are as follows:

```
ADD FIELD ORDER-RECTYPE OF ORDER POS 1 SIZE 1 -
   DBN 'OA'
ADD FIELD ORDER-IDENT SIZE 9 TYPE ZONED CODE -
   DBNAME 'OB'
ADD FIELD ORDER-NUMBER OF ORDER-IDENT POS 1 -
   SIZE 7 TYPE ZONED UNSIGNED CODE
ADD FIELD ORDER-CUST-NUMBER SIZE 4
TYPE PACKED CODE -
   EDIT '999B9B999' HEA ('PLACED BY', 'CUSTOMER') -
   DBN 'OC' REMARKS REFERENCE NUMBER FOR THE -
   CUSTOMER WHO PLACED THIS ORDER
ADD FIELD ORDER-WRITTEN-BY SIZE 9 -
   TYPE ZONED CODE EDIT '999B99B999' HEADING -
   ('WRITTEN BY', 'SALESPERSON') DBN 'OD' -
   REMARKS REFERENCE NUMBER FOR THE -
   SALESPERSON WHO WROTE THIS ORDER
ADD FIELD ORDER-DATA SIZE 13 TYPE MIXED DBN 'OE'
ADD FIELD ORDER-PLACEDATE OF ORDER-DATA POS 1 -
   SIZE 4 TYPE PACKED DATE 'YYMMDD'
   .
   .
   .
```

The systems staffs at our example site have told us that only fields named OA, OB, OC, OD, and OE are written for an order record, and so we will not define the other ORDER-ITEM file fields in this record. The headings for OC and OD are copied directly from the DD07 report, as are the edit masks for OB, OC and OD. The DBSR field names are used as the dbname values for all the fields.

Note, however, that OE is in fact a group, whose subfields are not defined to ADABAS/C. Our systems staff has given us the subfield definitions, which begin with the ORDER-PLACEDATE field. Since ORDER-PLACEDATE is not defined as a separate field to the DBMS, it has no dbname in the ADD FIELD command. Continuing, the field definition commands for the ITEM record would be:

```
ADD FIELD ITEM-RECTYPE OF ITEM POS 1 SIZE 1 -
   DBNAME 'OA'
ADD FIELD ITEM-ORDER-IDENT SIZE 9 TYPE ZONED CODE -
   EDIT '9999999B99' DBNAME 'OB'
ADD FIELD ITEM-ORDER-NUMBER OF ITEM-ORDER-IDENT -
   POS 1 SIZE 7 TYPE ZONED UNSIGNED CODE
ADD FIELD ITEM-LINE-NUM OF ITEM-ORDER-IDENT POS 8 -
   SIZE 2 TYPE ZONED CODE
ADD FIELD ITEM-CUST-NUMBER SIZE 4 -
   TYPE PACKED CODE
   EDIT '999B9B999' HEA ('PLACED BY', 'CUSTOMER') -
   DBN 'OC' REM REFERENCE NUMBER FOR THE CUSTOMER -
   WHO PLACED THIS ORDER
ADD FIELD ITEM-QUANTITY SIZE 4 TYPE BINARY DEC 3 -
   EDIT 'ZZZ,ZZ9.999' DBN 'OI' REMARKS AMOUNT OF -
   ITEM ORDERED: LITERS FOR LIQUIDS, KILOS FOR -
   POWDERS, AND UNITS FOR ALL OTHERS
ADD FIELD ITEM-NAME SIZE 10 DBN 'OJ'
ADD FIELD ITEM-PRICE SIZE 4 TYPE BINARY -
   DEC 2 EDIT -
   '$Z,ZZZ,ZZ9.99-' DBN 'OK' REMARKS UNIT PRICE -
   CHARGED TO CUSTOMER
```

Our systems staff has informed us that fields OA, OB, OC, OI, OJ, and OK are stored for each ITEM record. Again, the size, type, decimal-point, edit-mask, heading, and remarks are all taken for the DBSR and DD07 reports, and the ADABAS/C internal name is used as the dbname value. The field definition commands for the CUSTOMER and SALESPERSON records could be coded as follows:

```
ADD FIELD CUSTOMER-NUMBER OF CUSTOMER POS 1 -
   SIZE 7 TYPE ZONED CODE EDIT '999B9B999' DBN 'CA'
ADD FIELD CUSTOMER-NAME SIZE 20 DBN 'CB'
ADD FIELD CUSTOMER-PRI-SALESPERSON - SIZE 9 TYPE ZON -
   CODE EDIT '999B99B9999' HEA ('CUST. PRIMARY', -
   'SALESPERSON') DBN 'CC' REMARKS REFERENCE -
   NUMBER FOR THE CUSTOMER'S PRIMARY SALESPERSON
ADD FIELD SALESPERSON-NUMBER OF SALESPERSON -
   POS 1 SIZE 9 TYPE ZONED CODE -
   EDIT '999B99B9999' -
   DBN 'SA'
ADD FIELD SALESPERSON-NAME SIZE 20 DBNAME 'SB'
ADD FIELD REGIONS-COVERED SIZE 2 TYPE ZONED CODE -
   OCCURS 4 EQUATE (1 = 'ALASKA', 2 = 'HAWAII', -
   .
   .
   .
   50 = 'MAINE', 51 = 'WASHINGTON, DC') DBN 'SC'
ADD FIELD SALESPERSON-YEARLY-DATE SIZE 22 -
   TYPE MIXED OCCURS 5 DBNAME 'SD'
ADD FIELD SALESP-YEARLY-NUM-SALES OF -
```

```
      SALESPERSON-YEARLY-DATA POS 1 SIZE 2 TYPE -
      BINARY HEADING ('NUMBER OF', 'SALES') DBN 'SE'
   ADD FIELD SALESP-YEARLY-AMT-SALES OF -
      SALESPERSON-YEARLY-DATA POS 3 SIZE 4 TYPE -
      PACKED HEADING ('YEAR-TOTAL','SALES (AMT)') -
      DECIMAL 2 EDIT '$ZZZ,ZZ9.99' DBN 'SF'
   ADD FIELD SALESP-YEARLY-AWARD OF -
      SALESPERSON-YEARLY-DATA POS 7 SIZE 16 TYPE -
      MIXED HEADING ('END OF YEAR', 'AWARD') DBN 'SG'
```

The ADABAS/C CUSTOMERS and SALESPERSONS files are both single-record files, and so we have defined one record for each.

Note: In the DD07 listing that the SC field is defined to ADABAS/C as a "translation" field, which is somewhat like a MetaSuite "EQUATE" field. ADABAS/C translation is accomplished with an ADABAS/C table file. MetaSuite users may either define the ADABAS/C translation table file as a record in a MetaSuite (ADABAS/C) file and use it as a TABLE file in their programs, or they may simply copy the table file contents into their MetaSuite Dictionary as an EQUATE clause on the ADD FIELD command (as shown here).

The SC field is a multi-valued field, and so is defined with the OCCURS clause. The maximum number of occurrences that may be expected in the database is used as the OCCURS value. The SD field is a periodic group, and so it is also defined with an OCCURS clause.

4.6. Defining Indexes

Each ADABAS/C descriptor is defined in the MetaSuite Dictionary as an index by an ADD INDEX command. The syntax of the command depends on the descriptor type (simple, subdescriptor, etc.). Each ADABAS/C field that has the DE or UQ (descriptor or unique descriptor) characteristic should be defined to MetaSuite as an index as well as a field. Each subdescriptor or superdescriptor should be defined to MetaSuite as an index. The syntax of the ADD INDEX command is described twice below -- the first as used to define simple descriptors, and the second as used to describe sub- and super- descriptors.

Command for Simple Descriptors

```
ADD INDEX index-name BASED ON field-name -
      [DBNAME 'dbname']
```

Naming the Index

index-name

index-name is an arbitrary name for the index entity being defined to the MetaSuite Dictionary. Choose any unique, descriptive name.

Specifying the Index Sequence Field

BASED ON *field-name-sequence*

The BASED ON option names the descriptor field on which the index is sequenced. *field-name* is the MetaSuite field name for any ADABAS/C field which has the DE (descriptor) or UQ (Unique descriptor) characteristic. This field is specified with a DBNAME.

Specifying the Index Dbname

```
DBNAME 'dbname'
```

The *dbname* is optional. If specified, it must be the same as the dbname for the sequence field, i.e. the 2-character field name from the DBSR report.

Discussion

MetaSuite will assume that an index defined in this way is a simple descriptor. It will use the index dbname in search lists, and it will use the field dbname in the format lists. It will use the field type and size when formatting data for search processing, and it will use the same type and size when processing the data returned for the record by the DBMS.

Command: Sub- and Super-Descriptors

```
ADD INDEX index-name BASED ON record-name
      TYPE type SIZE size DBNAME 'dbname'
```

Naming the Index

index-name

Index-name is an arbitrary name for the index entity being defined to the MetaSuite Dictionary. Choose any unique, descriptive name.

Specifying the Indexed Record

BASED ON *record-name*

The BASED ON option specifies the name of the indexed record.

This must be the name chosen on the ADD RECORD command for this record.

Specifying the TYPE and SIZE

TYPE *type* SIZE *size*

Sub - and super-descriptors have a data type and size that does not (necessarily) correspond to any field in the record. Therefore, their type and size are defined for the index independently of the fields in the record.

A subdescriptor is formed from a portion of a field. Its type will be related to the type of the base field, and its size will be less than or equal to the size of the base field. Use the following chart to determine subdescriptor type:

If this is the base field type...	And this is the portion of the base field used for the subdescriptor...	Then this is the subdescriptor TYPE...
MIXED	Any	MIXED
ZONED	Anything except the last byte.	ZONED UNSIGNED
ZONED	Anything including the last byte	ZONED

If this is the base field type...	And this is the portion of the base field used for the subdescriptor...	Then this is the subdescriptor TYPE...
PACKED	Anything except the last byte	PACKED UNSIGNED
PACKED	Anything including the last byte	PACKED
BINARY	Any	BINARY

The size of the index is determined directly from the DBSR description.

A superdescriptor is formed from portions of several fields -- its type should always be defined as MIXED, and its size taken from the DBSR descriptor (it is the sum of the sizes of the pieces.).

Examples

Looking at our DBSR and DD07 reports, we now construct the index definitions for the four sample database records. They are:

```
ADD INDEX ORDER-NUMBER-IX -
   BASED ON ORDER SIZE 7 -
   TYPE ZONED UNSIGNED DBNAME 'ON'
ADD INDEX ORDER-CUST-NUMBER-IX -
   BASED ON ORDER-CUST-NUMBER
ADD INDEX ORDER-WRITTENIX BASED ON ORDER-WRITTEN-BY
   .
   .
   .
ADD INDEX ITEM-ORDER-NUMBER-IX -
   BASED ON ITEM SIZE 7 -
   TYPE ZONED UNSIGNED DBNAME 'ON'
ADD INDEX ITEM-ORDER-IDENT-IX -
   BASED ON ITEM-ORDER-IDENT
ADD INDEX ITEM-CUST-NUMBER-IX -
   BASED ON ITEM-CUST-NUMBER
ADD INDEX ITEM-NAME-IX BASED ON ITEM-NAME
   .
   .
   .
ADD INDEX CUSTOMER-NUMBER-IX -
   BASED ON CUSTOMER-NUMBER
ADD INDEX CUSTOMER-NAME-IX BASED ON CUSTOMER-NAME
ADD INDEX CUST-PRI-SALESPERSON-IX BASED ON -
   CUSTOMER-PRI-SALESPERSON
   .
   .
   .
ADD INDEX SALESPERSON-NUMBER-IX -
   BASED ON SALESPERSON-NUMBER
ADD INDEX SALESPERSON-NAME-IX -
   BASED ON SALESPERSON-NAME
```

Each of the 'DE' and 'UQ' fields is defined as an index. The only subdescriptor in this database is the 'ON' descriptor.

From the DBSR report we see that it is bytes 1 through 7 of the 'OB' field that are zoned. Thus, the index is TYPE ZONED UNSIGNED SIZE 7. Since this descriptor appears in ADABAS/C file 1, and since that file is defined to MetaSuite as the ITEM and ORDER records, and since 'OB' is maintained for both record types, the descriptor is defined twice: once for the ITEM record and once for the ORDER record.

4.7. Defining Links

When you deal with multiple MetaSuite (ADABAS/C) records, often you will encounter situations in which one MetaSuite (ADABAS/C) record is related to another. The relationship is recorded in the database by duplicating data. In our example database, for instance, the CUSTOMER-PRI-SALESPERSON field always will be a copy of some SALESPERSON-NUMBER field. A CUSTOMER record is thus related to a SALESPERSON record (specifically, to the one with the equivalent SALESPERSON-NUMBER value).

ADABAS/C Treatment of Relationships

Note that ADABAS/C itself does nothing to enforce this relationship; a CUSTOMER-PRI-SALESPERSON value may not correspond to any SALESPERSON-NUMBER value. The relationship is maintained solely by the programs that store and modify data in the database, and the existence and reliability of the relationship are the results of programming standards used at your site. You cannot go to any ADABAS/C report to discover the relationship; you must find it by becoming familiar with your shop standards for the database in question.

Defining Links to the MetaSuite Dictionary

Although the process of tracking down these relationships is not easy, it is worthwhile because the relationships add tremendous power to the MetaSuite database access capabilities. Therefore, we recommend that you define the relationships to your MetaSuite Dictionary. The relationships are defined as LINKS.

Each LINK definition relates two MetaSuite (ADABAS/C) records to one another. The "basis" of the link is a field in one record (the "from" record, so called because data is taken from that record and used to locate the other record), and a descriptor in the other (the "to" record, so called because the descriptor leads to that record).

There are (as with INDEX), two forms of the ADD LINK command. In this case, the first is the older command supported only for upwards compatibility; documentation is provided for maintenance purposes only. The second form is the newer form, and it should be used for all new definitions.

Command: Old Form

```
ADD LINK link-name
      FROM record-name-a
      TO record-name-b BASED ON field-name-b
```

Naming the Link

link-name

link-name is an arbitrary, unique name that you assign to identify this particular relationship.

Identifying the FROM Record

```
FROM record-name-a
```

record-name-a is the name of some MetaSuite (ADABAS/C) record that must be defined with a STORAGE-KEY. The STORAGE-KEY field will be assumed to be a simple descriptor (it must, therefore, be defined with a DBNAME). The storage-key field in record-name-a is used to obtain values that lead to the TO record.

Identifying the TO Record

```
TO record-name-b BASED ON field-name-b
```

record-name-b is the name of some other MetaSuite (ADABAS/C) record in the same MetaSuite file. *field-name-b* must be a field in record-name-b, and it is assumed also to be a simple descriptor (and, thus, also must be defined with a DBNAME). The field-name-b descriptor will be used to locate occurrences of record-name-b that are related to record-name-a.

Examples

By looking at our previous ADD RECORD and ADD FIELD commands and consulting with our systems staff, we can construct ADD LINK commands of the old type:

```
ADD LINK SALES-CUST FROM SALESPERSON TO CUSTOMER -
   BASED ON CUSTOMER-PRI-SALESPERSON
ADD LINK SALES-ORDER FROM SALESPERSON TO ORDER -
   BASED ON ORDER-WRITTEN-BY
ADD LINK CUST-ORDER FROM CUSTOMER TO ORDER -
   BASED ON ORDER-CUST-NUMBER
ADD LINK CUST-ITEM FROM CUSTOMER TO ITEM -
   BASED ON ITEM-CUST-NUMBER
```

In each case, the TO record contains a simple descriptor (named as the BASED ON value in each command) which, in turn, contains values that duplicate the STORAGE-KEY field value of the FROM record. Note again that this is the obsolete form of the ADD LINK command; these examples will not be repeated in this document.

Command: New Form

```
ADD LINK link-name -
      FROM record-name-a -
            BASED ON {field-name-a | index-name-a} -
      TO record-name-b -
            BASED ON {field-name-b | index-name-b}
```

Naming the Link

```
link-name
```

link-name is an arbitrary, unique name which you assign to identify this particular relationship.

Identifying the FROM Record Link

```
FROM record-name-a -
      BASED ON {field-name-a | index-name-a}
```

The FROM *record-name* BASED ON clause may specify a field-name or an index-name. The FROM record basis must be a field or a simple descriptor. If it specifies an index, that index definition must be of the "BASED ON *field-name*" type. The field that ultimately forms the basis for the link on the FROM side must be defined with a dbname, or it must be a subfield with a dbname. *index-name-a* must be an index based on a field within *record-name-a*, or *field-name-a* must be a field within *record-name-a*.

Identifying the TO Record Link Basis

```
TO record-name-b -
      BASED ON {field-name-b | index-name-b}
```

The TO *record-name* BASED ON clause may specify a field-name or an index-name. The TO record link basis must be a descriptor (of any kind). If it specifies a field, then that field must be defined with a dbname. If it specifies an index, the index may be of any type. *index-name-b* must be an index based on *record-name-b* (or in a field in that record), and *field-name-b* must be field in *record-name-b*.

Discussion

The link may be traversed in either "direction" if the basis for both records is a simple descriptor; otherwise, it must be traversed in the FROM-TO direction. Regardless of which way the link is traversed, the field dbname will be used to retrieve the link basis value for each occurrence of the higher-level record in the FIELD BUFFER. The INDEX dbname and the link basis value will be used together to retrieve the lower-level record in the FILE BUFFER.

Examples

By looking at our previous ADD RECORD, ADD FIELD and ADD INDEX commands, and consulting with our systems staff, we construct ADD LINK commands as follows:

```
ADD LINK SALES-CUST FROM SALESPERSON BASED ON -
   SALESPERSON-NUMBER-IX TO CUSTOMER BASED ON -
   CUST-PRI-SALESPERSON-IX
ADD LINK SALES-ORDER FROM SALESPERSON BASED ON -
   SALESPERSON-NUMBER-IX TO ORDER BASED ON -
   ORDER-WRITTENIX
ADD LINK CUST-ORDER FROM CUSTOMER BASED ON -
   CUSTOMER -NUMBER-IX TO ORDER BASED ON -
   ORDER-CUST-NUMBER-IX
ADD LINK CUST-ITEM FROM CUSTOMER BASED ON -
   CUSTOMER-NUMBER-IX TO ITEM BASED ON -
   ITEM-CUST-NUMBER-IX
ADD LINK ORDER-ITEM FROM ORDER BASED ON -
   ORDER-NUMBER -
   TO ITEM BASED ON ITEM-ORDER-NUMBER-IX
   ADD LINK ITEM-ORDER FROM ITEM BASED ON -
   ITEM-ORDER-NUMBER -
   TO ORDER BASED ON ORDER-NUMBER-IX
```

The first four links are all based on simple descriptors. Therefore, the BASED ON clauses in these first four could specify either the INDEX name or the FIELD name; we have chosen the index-names arbitrarily.

The last two links show the use of a non-descriptor field in the FROM record and a subdescriptor on the TO record. Note that the ORDER-ITEM and ITEM-ORDER are, for this reason, "one-way" links. The following MetaSuite program PATH definition uses ORDER-ITEM properly:

```
PATH DEFINITION FOR WIDGET-SALES-DB
WHERE : BUFFER (ORDER,ITEM VIA ORDER-ITEM)
```

Note: The "VIA ORDER-ITEM" is the default in this case, and therefore, could be omitted by the programmer. The following PATH definition uses ORDER-ITEM the wrong way and would produce errors during syntax checking:

```
PATH DEFINITION FOR WIDGET-SALES-DB
WHERE: BUFFER (ITEM,ORDER VIA ORDER-ITEM)
```

All other links are "two-way". The following two PATH definitions, though different, are both legal:

```
PATH DEFINITION FOR WIDGET-SALES-DB
WHERE: BUFFER (CUSTOMER,SALESPERSON VIA SALES-CUST)
PATH DEFINITION FOR WIDGET-SALES-DB
WHERE: BUFFER (SALESPERSON,CUSTOMER VIA SALES-CUST)
```

The rules for determining whether a link is "one-way" or "two-way" are:

- If the FROM record is BASED ON a non-descriptor field, then the link is one-way.
- If the TO record is BASED ON a sub- or super- descriptor, then the link is one-way.

All other links are two-way.

The rules for using one- and two-way links in PATH definitions are as follows:

- If a link is one-way, then the records in a PATH must be specified in FROM-TO sequence.
- If a link is two-way, then the records in a PATH may be specified in FROM-TO or TO-FROM sequence.

JCL Considerations

The MetaSuite Dictionary Maintenance program (CQSMAINT job) does not require any JCL modifications when defining ADABAS/C databases.

4.8. Definition Example

```
ADD FILE WIDGET-SALES-DB TYPE ADABAS
ADD RECORD ORDER SIZE 34 -
   KEY (ORDER-RECTYPE = '0') DBNAME '01'
ADD FIELD ORDER-RECTYPE POS 1 SIZE 1 DBN 'OA'
ADD FIELD ORDER-IDENT SIZE 9 TYPE ZONED CODE -
   DBNAME 'OB'
ADD FIELD ORDER-NUMBER OF ORDER-IDENT POS 1 -
   SIZE 7 TYPE ZONED UNSIGNED CODE
ADD FIELD ORDER-CUST-NUMBER SIZE 4 -
   TYPE PACKED CODE -
   EDIT '999B9B999' HEA('PLACED BY', 'CUSTOMER') -
   DBN 'OC' REMARKS REFERENCE NUMBER FOR THE -
   CUSTOMER WHO PLACED THIS ORDER
ADD FIELD ORDER-WRITTEN-BY SIZE 0 -
   TYPE ZONED CODE EDIT '999B99B9999' HEADING -
```

```
   ('WRITTEN BY', 'SALESPERSON') DBN 'OD' -
   REMARKS REFERENCE NUMBER FOR THE -
   SALESPERSON WHO WROTE THIS ORDER
ADD FIELD ORDER-DATA SIZE 13 TYPE MIXED DBN 'OE'
ADD FIELD ORDER-PLACEDATE OF ORDER-DATA POS 1
   SIZE 4 TYPE PACKED DATE 'YYMMDD'
ADD RECORD ITEM SIZE 30 -
   KEY (ITEM-RECTYPE = 'I') DBNAME '01'
ADD FIELD ITEM-RECTYPE POS 1 SIZE 1 DBNAME 'OA'
ADD FIELD ITEM-ORDER-IDENT SIZE 9 TYPE ZONED CODE -
   EDIT '9999999B99' DBNAME 'OB'
ADD FIELD ITEM-ORDER-NUMBER OF ITEM-ORDER-IDENT -
   POS 1 SIZE 7 TYPE ZONED UNSIGNED CODE
ADD FIELD ITEM-LINE-NUM OF ITEM-ORDER-IDENT POS 8 -
   SIZE 2 TYPE ZONED CODE
ADD FIELD ITEM-CUST-NUMBER SIZE 4 -
   TYPE PACKED CODE -
   EDIT '999B9B999' HEA ('PLACED BY', 'CUSTOMER') -
   DBN 'OC' REM REFERENCE NUMBER FOR THE CUSTOMER -
   WHO PLACED THIS ORDER
ADD FILE ITEM-QUANTITY SIZE 4 TYPE BINARY DEC 3 -
   EDIT 'ZZZ,ZZ9.999' DBN 'OI' REMARKS AMOUNT OF -
   ITEM ORDERED: LITERS FOR LIQUIDS, KILOS FOR -
   POWDERS, AND UNITS FOR ALL OTHERS
ADD FIELD ITEM-NAME SIZE 10 DBN 'OJ'
ADD FIELD ITEM-PRICE SIZE 4 TYPE BINARY -
   DEC 2 EDIT -
   '$Z,ZZZ,ZZ9.99-' DBN 'OK' REMARKS UNIT PRICE -
   CHARGED TO CUSTOMER
ADD RECORD CUSTOMER SIZE 36 -
   STORAGE-KEY CUSTOMER-NUMBER DBNAME '02'
ADD FIELD CUSTOMER-NUMBER POS 1 -
   SIZE 7 TYPE ZONED CODE EDIT '999B9B999' -
   DBN 'CA'
ADD FIELD CUSTOMER-NAME SIZE 20 DBN 'CB'
ADD FIELD CUSTOMER-PRI-SALESPERSON SIZE 9 -
   TYPE ZON -
   CODE EDIT '999B99B9999' HEA ('CUST. PRIMARY', -
   'SALESPERSON') DBN 'CC' REMARKS REFERENCE -
   NUMBER FOR THE CUSTOMER'S PRIMARY SALESPERSON
ADD RECORD SALESPERSON SIZE 148 -
   STORAGE-KEY SALESPERSON-NUMBER DBNAME '03'
ADD FIELD SALESPERSON-NUMBER POS 1 SIZE 9 -
   TYPE ZONED CODE EDIT '999B99B9999' DBN 'SA'
ADD FIELD SALESPERSON-NAME SIZE 20 DBNAME 'SB'
ADD FIELD REGIONS-COVERED SIZE 2 TYPE ZONED CODE -
   OCCURS 4 EQUATE (1 = 'ALASKA', 2 = 'HAWAII', -
   .
   .
   .
   50 = 'MAINE', 51 = 'WASHINGTON, DC') DBN 'SC'
ADD FIELD SALESPERSON-YEARLY-DATA SIZE 22 -
   TYPE MIXED OCCURS 5 DBNAME 'SC'
ADD FIELD SALESP-YEARLY-NUM-SALES OF -
   SALESPERSON-YEARLY-DATA POS 1 SIZE 2 TYPE -
   BINARY HEADING ('NUMBER OF', 'SALES') DBN 'SE'
ADD FIELD SALESP-YEARLY-AMT-SALES OF -
   SALESPERSON-YEARLY-DATA POS 3 SIZE 4 TYPE -
   PACKED HEADING ('YEAR-TOTAL','SALES (AMT)') -
```

```
   DECIMAL 2 EDIT '$ZZZ,ZZ9.99' DBN 'SF'
ADD FIELD SALESP-YEARLY-AWARD OF -
   SALESPERSON-YEARLY-DATA POS 7 SIZE 16 TYPE -
   MIXED HEADING ('END OF YEAR', 'AWARD') DBN 'SG'
ADD INDEX ORDER-NUMBER-IX BASED ON ORDER SIZE 7 -
   TYPE ZONED UNSIGNED DBNAME 'ON'
ADD INDEX ORDER-CUST-NUMBER-IX -
   BASED ON ORDER-CUST-NUMBER
ADD INDEX ORDER-WRITTENIX BASED ON ORDER-WRITTEN-BY
   .
   .
   .
ADD INDEX ITEM-ORDER-NUMBER-IX -
   BASED ON ITEM SIZE 7 -
   TYPE ZONED UNSIGNED DBNAME 'ON'
ADD INDEX ITEM-ORDER-IDENT-IX -
   BASED ON ITEM-ORDER-IDENT
   ADD INDEX ITEM-CUST-NUMBER-IX -
   BASED ON ITEM-CUST-NUMBER
ADD INDEX ITEM-NAME-IX BASED ON ITEM-NAME
   .
   .
   .
ADD INDEX CUSTOMER-NUMBER-IX -
   BASED ON CUSTOMER-NUMBER
ADD INDEX CUSTOMER-NAME-IX BASED ON CUSTOMER-NAME
ADD INDEX CUST-PRI-SALESPERSON-IX BASED ON -
   CUSTOMER-PRI-SALESPERSON
   .
   .
   .
ADD INDEX SALESPERSON-NUMBER-IX -
   BASED ON SALESPERSON-NUMBER
ADD INDEX SALESPERSON-NAME-IX -
   BASED ON SALESPERSON-NAME
ADD LINK SALES-CUST FROM SALESPERSON BASED ON -
   SALESPERSON-NUMBER-IX TO CUSTOMER BASED ON -
   CUST-PRI-SALESPERSON-IX
ADD LINK SALES-ORDER FROM SALESPERSON BASED ON -
   SALESPERSON-NUMBER-IX TO ORDER BASED ON -
   ORDER-WRITTENIX
ADD LINK CUST-ORDER FROM CUSTOMER BASED ON -
   CUSTOMER-NUMBER-IX TO ORDER BASED ON -
   ORDER-CUST-NUMBER-IX
ADD LINK CUST-ITEM FROM CUSTOMER BASED ON -
   CUSTOMER-NUMBER-IX TO ITEM BASED ON -
   ITEM-CUST-NUMBER-IX
ADD LINK ORDER-ITEM FROM ORDER -
   BASED ON ORDER-NUMBER -
   TO ITEM BASED ON ITEM-ORDER-NUMBER-IX
ADD LINK ITEM-ORDER FROM ITEM -
   BASED ON ITEM-ORDER-NUMBER -
   TO ORDER BASED ON ORDER-NUMBER-IX
```

CHAPTER 5
Program Generation

As with any other DBMS system, the use of ADABAS/C files requires that the MetaSuite user know and understand the relationships between the various types of data in the ADABAS/C files.

This chapter describes the ways in which those definitions are used to construct applications that will access the information stored in the database. In addition, users must be aware of the fact that there are many ways to access the information stored in the database, some more efficient than others. If efficiency is a consideration, you can obtain a more thorough knowledge of the database from the systems staff at your site.

As above, the term "ADABAS/C file" will be used to refer to any files maintained by the DBMS. The term "MetaSuite (ADABAS/C) file" will be used to refer to a MetaSuite file definition whose type is "ADABAS/C". Remember that a MetaSuite (ADABAS/C) file will usually correspond to a group of ADABAS/C files, and that a MetaSuite record will usually correspond to an ADABAS/C file.

All commands not affected by the use of MetaSuite (ADABAS/C) files are described in the MetaSuite User and Reference Guides.

Topics covered:

5.1. Summary of Definitions Affected

This section summarizes which file definitions and procedural commands are affected by the use of MetaSuite ADABAS/C files, and which are not.

Data Source Definitions

Of the two data source definitions, the FIELD definition is not affected by the use of ADABAS/C database files. The file definitions (Automatic File, Match File, Control File, Controlled By File) identifies a particular MetaSuite (ADABAS/C) file to be processed. Using file definitions with MetaSuite (ADABAS/C) files is described below.

Note: The NEW, TABLE, ASSIGN, and ENCRYPT options are not allowed for ADABAS/C files.

Formatting Definitions

The report and transfer specifications (REPORT, TITLE, STRATIFY, etc.) are not affected in any way by the use of MetaSuite (ADABAS/C) files.

Procedural Commands

The syntax of the procedural commands is not affected by the use of MetaSuite (ADABAS/C) files, but the EXCLUDE and the GET commands are much more powerful when used with MetaSuite (ADABAS/C) files. ADABAS/C considerations when using the EXCLUDE, START, and GET commands are discussed later in this chapter.

5.2. Defining Files

```
AUTOMATIC FILE, CONTROLLED FILE, CONTROLLED BY FILE,
and MATCH FILE Definitions:
   NAME:                file-name
   PREFIX:              prefix-value
   MATCH FIELD:         match-key
   CONTROLLED BY FILE:  controlling-file-name
   CONTROL KEY          control-key
   DBNAME:              workfield-name

PATH Definition:
   WHERE: BUFFER (buffer-specification)]
```

Identifying the File

file-name

file-name is the name of a MetaSuite (ADABAS/C) file defined to the MetaSuite Dictionary.

Assigning a Prefix

PREFIX: *prefix-value*

The PREFIX option is used to provide unique file, record, and field names when accessing different versions of the same file, or to provide unique names when a record is defined in two or more files accessed in the program. *prefix-value* must be four characters in length, and must begin with an alphabetic character.

Specifying a DBNAME Workfield

The *DBNAME* option is used to identify a workfield that must have been defined in a previous FIELD definition. The workfield must be defined as TYPE MIXED SIZE 8. The value of the workfield will be used to fill in the ADDITIONS 3 field in the ADABAS/C user control block prior to each call to the database management system -- this field is used to hold the ADABAS/C password which will provide access to the ADABAS/C files. The use of the ADDITIONS 3 field is documented in the ADABAS/C Command Reference Manual ((c) Software AG of North America). The MetaSuite program workfield may be initialized either to spaces (the default) or to the 8-character password for the various ADABAS/C files to which this MetaSuite FILE definition provides access. The value of the workfield may be set as a runtime parameter, but the value must not be altered by the procedural code in your program.

If the workfield is set to spaces when the program begins execution, or if the DBNAME option is not specified in the FILE definition, then no ADABAS/C password will be used in database requests. If the DBNAME option is specified and a non-blank password is placed in the workfield (either using the INITIAL option of the FIELD definition or using the field as a runtime parameter), then that password value will be passed to ADABAS/C in every database request for this MetaSuite (ADABAS/C) file.

Identifying Match Keys

```
MATCH FIELD: match-key(s)
```

In a MATCH FILE definition, the MATCH FIELD option functions exactly as it does with non-ADABAS/C files.

Controlled Retrieval

Controlled retrieval of database records is accomplished through the use of the CONTROLLED FILE and CONTROLLED BY FILE definitions in the same way that it is for non-database files. However, the most powerful application of the CONTROLLED BY is with database files. Refer to the *Controlled File* and *Controlled By File* dialog boxes in the *MetaStore Manager User Guide* for a complete discussion of CONTROLLED and CONTROLLED BY file definitions.

Controlling Access Through Procedural Commands

To access an ADABAS/C file through the use of procedural GET commands, define the file as a Controlled file. A MetaSuite (ADABAS/C) file must be either defined as a Controlled file or a Path Buffer must be supplied. Optionally, a Controlled file may be buffered.

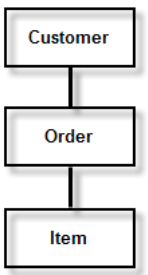
Combining Physical Records

```
PATH DEFINITION
   BUFFER (buffer-specification)
```

A MetaSuite (ADABAS/C) file must either be defined as a Controlled file or a Path BUFFER statement must be supplied for it. A BUFFER statement simplifies the processing of a database by allowing the user to view data from multiple record types as a single unit of data. This process of collecting data from multiple records to be treated as a single unit of data is sometimes called "file flattening". A BUFFER statement identifies the record types to be processed, and expresses the relationships between the records in hierarchical terms. The records named in the *buffer specification* are said to be either "path records", or "associated records". Link relationships are used to move from one occurrence of a record in the buffer to another, and are used to move from one record type in the buffer hierarchy to another.

Path Records

A "path" is a continuous hierarchical route through the records of the database. For example, we might define a path through our example database as beginning at CUSTOMER, and proceeding to ORDER, and from there to ITEM; each of these three records would be a "path record". This type of structure might be diagrammed as follows:

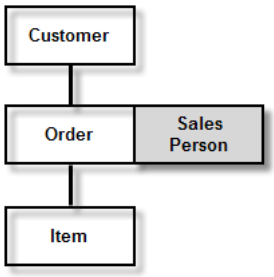


The system would process this path by obtaining the first CUSTOMER record, obtaining the first ORDER record for that CUSTOMER, and then obtaining the first ITEM record for that ORDER. This collection of data would then be available to the MetaSuite program procedures as the first buffer of data. The system would then attempt to obtain another ITEM record for the same ORDER record, and if successful, would return the new ITEM record along with the old CUSTOMER and ORDER records as the next buffer of data. When there are no more ITEM records for an ORDER, the next ORDER record for the current CUSTOMER record will be obtained, along with the first ITEM record for that order. This process continues until there are no more ORDER and ITEM records for the first CUSTOMER, at which point the system will obtain the next CUSTOMER record along with its first ORDER and ITEM records.

Note: A DBMS link relationship must exist between each path record and the next "lower" path record in the hierarchy.

Associated Records

It will sometimes be necessary to process records that are related to a path record, but which do not directly participate in the hierarchy. Referring once again to our example database, assume that we need to include some information from the SALESPERSON record that is related to each ORDER record. The database hierarchy we wish to process is the same, but we need some additional data at the middle level. This construct could be diagrammed:



In this construct, the SALESPERSON record is defined as an "associated record" in the MetaSuite buffer specification. The system would process the main path as described in the previous example, but each time an ORDER record was obtained, the related SALESPERSON record would also be obtained.

Note: The user can request either a single occurrence or multiple occurrences of an associated record type in the buffer. A DBMS link relationship must exist between each associated record and a path record. Associated record occurrences are linked to path records, but do not participate in the hierarchical path.

When used with a MetaSuite (ADABAS/C) file, the BUFFER statement specifies both the particular database record types o f interest, and the navigation path that will be used to retrieve them. The full syntax of the BUFFER statement when used with MetaSuite (ADABAS/C) files is described below.

BUFFER Statement Syntax

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

Identifying the Entry Record and Access Technique

entry-record [VIA *index-name*]

entry-record may be any record defined in the file. The traverse through the database will be based on this record type. If the entry record is qualified by the VIA option, the entry records will be retrieved in sequence by the index specified by *index-name*. Otherwise, they will be retrieved sequentially from the database in the order in which they are stored.

Identifying the Subordinate Records

subordinate-record

subordinate-record names another record in the database.

There may be up to 15 subordinate records specified following the entry record. A subordinate record may be any record that can be retrieved by traversing link relations in the database. For that reason, every subordinate record must be related by a link to some previously-named record in the BUFFER statement.

Note: The designation "subordinate" does not necessarily correspond to a hierarchical relationship within the database, but rather to the hierarchy set up in the MetaSuite buffer.

Identifying the Relationships between Records

[VIA {*related-record* | *link-name*}]

The VIA option explicitly names the relationship between the subordinate record and some other record previously named in the buffer specification (the higher-level record). *related-record* names the higher-level record, or *link-name* names the link between the named subordinate record and the higher-level record. If this option is omitted, it is assumed that the named subordinate record is related to the preceding path record in the buffer statement. Note that some links are "one-way" (see previous section of this document). If you specify VIA link-name using a one-way link, then the subordinate-record must be the TO record in the link definition.

Identifying Associated Records

OCCURS *number-times*

The *OCCURS* option is used to indicated to MetaSuite that a record is to be an associated record, i.e., not a path record. The maximum number of occurrences that can be specified is 99. This option cannot be coded for path records.

Examples

The following examples illustrate the use of buffer processing. All are based on the example database defined earlier. Both single-path and multiple-path buffers are illustrated.

Single-Path Example

```
PATH DEFINITION FOR WIDGET-SALES-DB
   WHERE: BUFFER (CUSTOMER, ORDER)
```

This PATH definition will cause the CUSTOMER records to be retrieved from the WIDGET-SALES database in sequential order. Each CUSTOMER will appear several times in the input stream, each time with another of its related ORDER records. A report containing only the following detail line:

```
   REPORT DETAIL DEFINITION FOR DETAIL-1
   REPORT VALUE: CUSTOMER-NUMBER, ORDER-NUMBER
```

```
REPORT VALUE DEFINITION FOR CUSTOMER-NUMBER
x SHORT
```

would print the following:

```
CUSTOMER    ORDER
NUMBER      NUMBER
*********   *******
123 6 204   2948 22
            2970 37
            2948 11
154 6 207   1648 21
            1750 57
158 3 441   3187 20
    .
    .
    .
```

Inverted Path Hierarchy Example

```
PATH DEFINITION FOR WIDGET-SALES-DB
WHERE: BUFFER (ORDER, CUSTOMER)
```

This PATH definition will return exactly the same information as the previous one, except that the input data will be in the storage order of the ORDER records, instead of CUSTOMER order. Depending on other program functions, such as sorting and record selection, relative record population sizes and densities, and DBMS internal configuration, one or the other method will be more efficient. If efficiency is a consideration, consult with your systems staff for advice.

Note: CUST-ORDER, the link that MetaSuite automatically uses in this example, is two-way. If it were one-way, then this buffer would not be legal.

Index-Sequential Access Example

```
PATH DEFINITION FOR WIDGET-SALES-DB
WHERE: BUFFER (CUSTOMER VIA CUST-NAME-IX, ORDER)
```

This PATH definition also returns the same information as the first example. As in the first example, the input buffers will be grouped by customer, but the customer records will be accessed in sequence by name. If the program reports required the data to be sorted in customer name order, use of this PATH definition would eliminate the need for a file sort.

Associated Record Example

```
PATH DEFINITION FOR WIDGET-SALES-DB
WHERE: BUFFER (SALESPERSON, CUSTOMER OCCURS 5, ORDER)
```

In this buffer specification, CUSTOMER records are assumed by MetaSuite to be related to SALESPERSON records; the link used will be SALES-CUSTOMERS. Since CUSTOMER is an associated record (the OCCURS clause specifies that), ORDER records will also be assumed to be related to SALESPERSON records; the link used for that relationship will be SALES-ORDERS. This statement will construct a buffer with the following layout:

```
SALESPERSON
CUSTOMER(1)
CUSTOMER(2)
CUSTOMER(3)
CUSTOMER(4)
CUSTOMER(5)
```

The first input buffer will contain the first SALESPERSON record stored in the database, the first five of its CUSTOMER records, and its first ORDER record. The next input buffer will contain the same SALESPERSON and CUSTOMER data, and the next ORDER belonging to this SALESPERSON. This will continue until the end of the set of ORDERs owned by SALESPERSON are reached, at which point the input buffer will contain a new SALESPERSON, its first five CUSTOMER records, and its first ORDER. Note that there may be fewer than five CUSTOMER records for a particular SALESPERSON. The system-defined workfield CUSTOMER SYS-BUFFER-COUNT always indicates the number of CUSTOMER records actually present in the current buffer. Refer to the *MetaMap Manager User Guide* for a description of the SYS-BUFFER-COUNT system-defined workfield. Note that all of the statements shown above would have appeared in the MetaSuite program listing with a Buffer Path Analysis Report produced by the program generator. The path analysis report produced for this example file would be:

```
PATH  PATH RECORDS  ASSOCIATED RECORDS
****  ************  ******************
   1  SALESPERSON   CUSTOMER(01 TO 05)
      ORDER
```

This path analysis report shows that successive input buffers contain the hierarchy: SALESPERSON, ORDER. It also shows that up to five CUSTOMER records are available for each SALESPERSON in the buffer. Note that fields in the CUSTOMER records in this buffer would always be referenced with a subscript, e.g. CUSTOMER-NAME(1) or CUSTOMER-NAME(WORK-SUB).

Multiple-Path Buffer Example

```
PATH DEFINITION FOR WIDGET-SALES-DB
WHERE: BUFFER (SALESPERSON,CUSTOMER,ORDER VIA SALESPERSON)
```

This buffer is an example of a multiple-path buffer. The CUSTOMER record is assumed to be related to the SALESPERSON record, forming the first path and the ORDER record is explicitly related to SALESPERSON (because of the VIA clause), forming the second path. The path analysis report for this file would be as follows:

```
PATH  PATH RECORDS  ASSOCIATED RECORDS
****  ************ ******************
   1  SALESPERSON
      CUSTOMER
   2  SALESPERSON
      ORDER
```

The buffer structure that would be constructed for this file is as follows:

```
SALESPERSON
CUSTOMER
ORDER
```

During execution, this buffer will always contain a SALESPERSON record, and may contain a new occurrence of either a CUSTOMER record or an ORDER record, but not a new occurrence of both. The system-defined workfields CUSTOMER SYS-BUFFER-COUNT and ORDER SYS-BUFFER-COUNT indicate which (if either) is present. A program that printed the SALESPERSON-NUMBER, CUSTOMER-NUMBER and ORDER-NUMBER values for this buffer only when those values changed would produce the following output:

```
SALESPERSON  CUSTOMER   ORDER
NUMBER       NUMBER     NUMBER
***********  ********   *******
524 37 0006  123 6 204
                        298 22
                        2970 37
                        2948 11
                        23731 1149
```

```
                126 7 311
                185 2 444
                154 6 207
                                1648 21
                                1750 57
445 22 2039     938 9 444
                                .
                                .
                                .
```

Note: All of the CUSTOMER records related to a given SALESPERSON record are retrieved and used to construct successive buffers, and then the ORDER records for the SALESPERSON are retrieved. The sequence of records used to build the successive buffers is the same as the sequence specified in the BUFFER statement.

Whenever you set up a buffered file, the structure and associations represented by your buffer specifications are displayed by the MetaSuite program generator in the form of a buffer path analysis report, as shown above. Buffer specifications can be arbitrarily complex, although there is a limit of 15 paths and 16 records per buffer. It is up to you to determine what action you want your program to take depending on the actual contents of each input buffer. You always determine the buffer contents by testing the appropriate SYS-BUFFER-COUNT workfields.

Note: MetaSuite (ADABAS/C) files may be buffered and controlled at the same time (as opposed to regular files that may be one or the other, but not both), and in fact, must be either buffered or controlled. When you define as file as being both Controlled and buffered, the GET command that you place in your procedural code specifies how to retrieve the entry record. The rest of the buffer is filled according to the rules described above. Each GET command will cause a new entry record to be retrieved and the entire buffer to be refilled.

Controlling Access through Key Values on Another File

`CONTROLLED BY FILE: controlling-file-name`
`CONTROL KEY control-key`

The CONTROLLED BY FILE option indicates that the records in one file will be accessed based on the information stored in, or derived from, another (controlling) file. This information (*control-key*) can be either a field in the controlling file or a workfield whose value is determined in the input procedure for the controlling file.

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

A file that is Controlled By another file cannot be defined as a Match File. Furthermore, the File Initial procedure for the Controlled By file cannot include commands that require a second pass: that is, the File Initial procedure cannot request a SORT, EXTRACT, or PRE-PASS.

The FILE definition for the controlling file (controlling-file-name) must come before the FILE definition for the Controlled file in the program. Also, the controlling file cannot itself be Controlled, although it can be Controlled By another file. (You can nest Controlled By specifications up to a maximum of 20 files.)

The Controlled file must be buffered. The Controlling file may be buffered, but if so, the buffer must define a single path. Either buffer can include associated records, however; that is, records specified with the OCCURS option. If the Controlling file is another MetaSuite (ADABAS/C) file, it must be buffered.

Note: If the control-key is a field in the Controlling file and the Controlling file is buffered, the named field must be the lowest level of the buffer hierarchy; that is, it must be the last non-OCCURS record defined for the path, or an OCCURS 1 record defined following the last non-OCCURS record.

Controlled By Example

Assume that we have a file called CUSTOMER-CONTROL that contains CUSTOMER-NUMBER-CONTROL values. We have another file, WIDGET-SALES-DB (the WIDGET-SALES-DB database) that contains CUSTOMER, ORDER, and ITEM records. The CUSTOMER record STORAGE-KEY is the CUSTOMER-NUMBER. We want to write a program that prints the 1985-order information for customers specified in the control file. The program is:

```
FILE DEFINITION FOR CUSTOMER-CONTROL

FILE DEFINITION FOR WIDGET-SALES-DB
CONTROLLED BY FILE: CUSTOMER-CONTROL -
CONTROL-KEY: CUSTOMER-NUMBER-CONTROL
PATH DEFINITION FOR WIDGET-SALES-DB
WHERE: BUFFER (CUSTOMER, ORDER, ITEM)

REPORT DEFINITION FOR REPORT-1
   .
   .
   .
RECORD INPUT PROCEDURE FOR ORDER
     IF ORDER-PLACEDATE LT 850101 EXCLUDE
     BEGIN REPORT 1 INPUT
     IF CUSTOMER SYS-BUFFER-COUNT EQ 0 -
          PRINT (bad control-key value detail line)
          EXIT
     IF ORDER SYS-BUFFER-COUNT EQ 0 -
          PRINT (no order data detail line)
          EXIT
     IF ITEM SYS-BUFFER-COUNT EQ 0 -
          PRINT (no item data detail line)
          EXIT
     PRINT (full data detail-line)
```

The CUSTOMER-CONTROL file will be read sequentially. The WIDGET-SALES-DB file will be entered using direct access, and successive buffers will be filled using sequential access.

The controlled sets available for report processing would be:

```
CONTROL     CUSTOMER     ORDER     ITEM
1620921286  1620921286   SC20221   CCC11111
                                   DDD22221
                                   DDD22225
                         SC20344   CCC11233
        .
        .
        .
        .
```

```
                All ORDER and ITEM data for CUSTOMER
                1620921286. If there were more CUSTOMER
                records with the same key value, they would
                follow with all their ORDER and ITEM data.
  2248374765    2248374765    SC41532    CCC22244
                .
                .
                .
```

Note: The CUSTOMER-CONTROL records could be in any sequence; they do not need to be sorted. The Record Input procedure uses the EXCLUDE command to eliminate any non-1985 order data, and all other controlled sets are passed to report processing. The use of EXCLUDE speeds up processing, in that no ITEM records are read for an excluded ORDER record.

The Report Input procedure checks for the following conditions:

- Missing CUSTOMER record (meaning that the CUSTOMER-CONTROL data did not correspond to an actual database record)
- Missing ORDER record (which occurs for a customer with no stored order information)
- Missing ITEM record (which occurs for an order with no line-items -- probably an exception condition)
- Complete information.

File Definition Combinations

The following table summarizes the various combinations of options available for the various FILE types. An "X" indicates that the particular option has been specified, and "O" indicates that it has not been specified. The RESULT of the particular combination of options is also given.

Match	Controlled	Controlled By	Buffered	Result
X	X	O	O	Not Allowed
X	O	O	O	Not Allowed
X	O	O	X	Works Fine (be careful which buffer record contains the match-key)
X	O	O	O	Not Allowed (Buffered or Controlled required)
O	X	X	O	Not Allowed
O	X	O	X	Works Fine
O	O	X	X	Works Fine
O	O	X	O	Not Allowed (Buffered required)
O	O	O	X	Works Fine.
O	O	O	O	Not Allowed (Buffered or Controlled required)

5.3. Reducing Processor Time and I/O Activity with EXCLUDE

When processing ADABAS/C database files, you can obtain a large saving in processor time and I/O activity by using the EXCLUDE and START commands. The START command is discussed later in this chapter.

If the file is buffered, then the EXCLUDE command when used within a Record Input Procedure allows you to bypass the accessing and processing of lower-level records in a buffer hierarchy, and to bypass the building of unwanted buffers.

As a slightly less efficient alternative, the EXCLUDE command can be used within a file input procedure with the record-name option.

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

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

Command

`EXCLUDE [`*`record-name`* `|` *`file-name`*`]`

The syntax for the EXCLUDE command is exactly the same for MetaSuite (ADABAS/C) files as it is for regular files. Each form of the command is discussed separately below.

Basic Example

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

```
FILE DEFINITION FOR WIDGET-SALES-DB
PATH DEFINITION
WHERE: BUFFER (CUSTOMER, BILL-TRANS)
REPORT DEFINITION FOR REPORT-1
REPORT DETAIL DEFINITION FOR DETAIL-1
REPORT VALUE: CUSTOMER-NUMBER, BILL-NUMBER
REPORT VALUE DEFINITION FOR CUSTOMER-NUMBER
x SHORT
```

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

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

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

Bypassing an Individual Buffer

EXCLUDE

To bypass (exclude) an individual buffer of records, simply use the EXCLUDE command as you would with a non-TOTAL file. The excluded buffer will not be processed. To select only buffers in which the BILL-NUMBER is less than 3500, you might add the following code to the basic example:

```
INPUT PROCEDURE FOR REPORT-1
IF BILL-NUMBER LT 3500 EXCLUDE
```

The report would contain information from buffers 3, 5, 6 and 9. Buffers 1, 2, 4, 7, and 8 would be excluded.

Bypassing Unwanted Buffers and Subordinate Records

EXCLUDE

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

Syntax and use of the Report Input procedure is described in the *MetaStore Manager User Guide*. Briefly, a Record Input Procedure is processed whenever the named record is accessed, before the completed buffer is processed by any file, report, or transfer procedure. A Record Input Procedure is coded immediately following any file procedures for the file to which the record is defined. The GET, COMPUTE, PRINT, and PUT commands cannot be used in a Record Input procedure.

Example

Given the basic example and buffers above, let's write a procedure to select (in the most efficient way) only customers with account numbers beginning with the digit 2. To do this, we'd add the following code to the basic example:

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

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

The same report results could have been obtained by using the same IF command in a Report or File Input procedure.

However, this would result in accessing all the records shown in the basic example and building all the buffers. This is dramatically less efficient than using a Record Input procedure for the same processing.

Bypassing Unwanted Buffers

EXCLUDE *record-name*

To bypass building unwanted buffers based on information in a completed buffer, use the EXCLUDE *record-name* command in a File Input procedure. This excludes the current buffer, and avoids building more buffers for the excluded record.

The difference between this technique and the previous one (excluding the record in a Record Input procedure) is that here the buffer is not completed if the record is excluded (i.e., no subordinate records are read), while with the Record Input approach, the buffer is completed before the exclusion. In both cases, no further buffers are built for the excluded record.

Example

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

```
FILE INPUT PROCEDURE FOR FILE WIDGET-SALES-DB
IF CUSTOMER-NUMBER NI (2000000 TO 2999999) EXCLUDE
CUSTOMER
```

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

Bypassing Controlled By Records

`EXCLUDE` *`file-name`*

EXCLUDE *file-name* is applicable only in file Input procedures. The file-name option is used when one file is CONTROLLED BY another to identify the controlling file data in which you are no longer interested. file-name must be the name of a controlling file (defined through the CONTROLLED BY option in the FILE definition).

Example

Assume that a program contained the following statements:

```
FILE DEFINITION FOR CUSTOMER-CONTROL
CONTROLLED FILE DEFINITION FOR WIDGET-SALES-DB
CONTROLLED BY CUSTOMER-CONTROL
CONTROL-KEY: CUSTOMER-NUMBER-CONTROL
PATH DEFINITION
WHERE: BUFFER (CUSTOMER,ORDER,SALESPERSON)
REPORT DEFINITION FOR REPORT-1
REPORT DETAIL DEFINITION FOR DETAIL-1
REPORT VALUE: CUSTOMER-NUMBER, ORDER-NUMBER,
SALESPERSON-NAME
REPORT VALUE DEFINITION FOR CUSTOMER-NUMBER
x SHORT
```

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

```
CUST-NUMBER        ORDER-NUMBER        SALESPERSON NAME     (controlled    set-
                                                            number)

***********        *******             ************

1620921286         SC20221             JONES                (1)
```

```
                SC20344          BLACK          (2)
                SC39374          REYES          (3)
2073849495      SC49483          CHANG          (4)
                SC25342          SMITH          (5)
                SC47365          KELLY          (6)
2994301234      SC36254          GABLE          (7)
                SC41092          HERON          (8)
3033009281      SC20982          HAMON          (9)
```

Nine controlled sets were constructed from the 22 ADABAS/C database records that were accessed (four CUSTOMERs, nine ORDERs and nine SALESPERSONs). If you wanted to exclude customer numbers in the range 2500000000 to 4000000000, you would add the following procedural commands to the program:

```
FILE INPUT PROCEDURE FOR WIDGET-SALES-DB
IF CUSTOMER-NUMBER IR (2500000000 TO 4000000000) -
   EXCLUDE CUSTOMER-CONTROL
```

The WIDGET-SALES-DB FILE INPUT procedure checks the customer number value in the CUSTOMER record, and if it is not the desired range, it excludes the higher level file. This causes the higher level file processing to retrieve the next CUSTOMER-CONTROL record, which in turn will cause the lower-level file processing to retrieve a new CUSTOMER record, plus its first ORDER and the SALESPERSON record for that ORDER.

This facility allows you to exclude unwanted controlled sets based on data in any portion of the controlled set.

The report output would consist of the same as the above except that it would include only control sets 1 through 6.

5.4. Reducing Processor Time and I/O Activity with START

The START command is used to bypass the accessing and processing of unwanted records or buffers. The syntax for the START command is exactly the same for MetaSuite (ADABAS/C) files as it is for regular files. It can only be used, however, with PATH definitions which specify the VIA index-name option.

Command

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

Identifying the Record or File to be Started

{record-name | file-name}

You can specify either the entry *record-name* or the *file-name* when the START command is used with a MetaSuite (ADABAS/C) file.

Specifying the Starting Position

`KEY = start-key`

The KEY option is used to specify an index key value less than or equal to the key value of the first record to be processed. The value specified can be a literal or the name of a field of the same general data type (alphabetic or numeric) as the index base field.

Example

When processing the same database as was used in the prior example, assume that you desire to process only those customers whose account numbers begin with the digits 285 or higher. You could code the program as follows:

```
FILE DEFINITION FOR WIDGET-SALES-DB
PATH DEFINITION
WHERE: BUFFER (CUSTOMER VIA CUST-NUMBER-IX, ORDER)
REPORT DEFINITION FOR REPORT-1
REPORT DETAIL DEFINITION FOR DETAIL-1
REPORT VALUE: CUSTOMER-NUMBER, ORDER-NUMBER
REPORT VALUE DEFINITION FOR CUSTOMER-NUMBER
x SHORT
FILE INITIAL PROCEDURE FOR WIDGET-SALES-DB
START WIDGET-SALES-DB KEY = 2850000
```

The report produced would contain only the data shown in the seventh, eighth and ninth buffers shown above. By coding the START command in the File Initial procedure, the database has been "positioned" at the first customer whose account number is greater than or equal to the specified value prior to accessing any of the entry records. Note that the BUFFER statement in this example differs from the prior example in that the VIA index-name clause has been added to the specification.

START Command Caution

It is very easy to put your program into an infinite loop through improper use of the START command. See the description of the START command in the *MetaStore Manager User Guide* for a discussion of the more common pitfalls when using the START command.

5.5. Controlled Access

Controlled access, either random or sequential, to MetaSuite (ADABAS/C) files is accomplished in exactly the same manner as controlled access to any standard non-database file, i.e., by defining the file as a Controlled File and using the procedural GET command, or by the CONTROLLED BY option in the File Definition. Controlled and Controlled By files are described in the *MetaStore Manager User Guide*, and the GET command syntax is described in the *MetaStore Manager User Guide*. Note that the success or failure of a GET command can be determined by checking the contents of the system-defined field, file-name SYS-IO-STATUS. For information about the use of the file-name SYS-IO-STATUS field, refer to the *MetaStore Manager User Guide*. A discussion of ADABAS/C considerations when using the GET command follows.

Command

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

The syntax for the GET command is exactly the same for MetaSuite (ADABAS/C) files as it is for regular files, but the choice of parameters to be used with the GET command depends on the presence or absence of the BUFFER option in the PATH definition for the file. Note that when controlled access is performed for a buffered file, the GET command not only retrieves the entry record, but also obtains any subordinate records necessary to complete the buffer.

Identifying the Records to be Read

```
{record-name | file-name}
```

If no PATH buffer is specified for the file, each GET command for the file must specify the *record-name* of the record to be obtained.

If a PATH buffer is defined, then either the *file-name* or the entry record name for the BUFFER option for that file can be specified.

Random Access

```
KEY = keyfield-value
```

The KEY option is used to specify a literal or the name of a field containing the retrieval key value to be used for the access. The specified *keyfield-value* must be of the same general data type (alphanumeric or numeric) as the retrieval key of the record to be obtained. For MetaSuite (ADABAS/C) files the retrieval key may be either a storage-keyfield or an index relationship basis field. When the entry record named in the BUFFER statement is specified with the VIA index-name option, the retrieval key is the index relationship basis field. In all other cases, the retrieval key is the storage-keyfield.

The following table summarizes the processing performed by the system for each combination of FILE and GET options that may be coded.

KEY	BUF	VIA	Processing Action
no	no	no	The next occurrence of the named record, as stored in the database, is retrieved.
no	yes	no	The next occurrence of the entry record, as stored in the database, is retrieved, and the buffer is refilled.
no	yes	yes	The next occurrence of the entry record, in index sequence, is retrieved, and the buffer is refilled.
yes	no	no	The first occurrence of the named record containing the keyfield-value in its storage-keyfield is retrieved.
yes	yes	no	The entry record occurrence containing the keyfield-value in its storage-keyfield is retrieved, and the buffer is refilled.
yes	yes	yes	The entry record occurrence containing the keyfield-value in the index relationship basis field is retrieved, and the buffer is refilled.

Example: Random Access Using Storage-Key

```
CONTROLLED FILE DEFINITION FOR WIDGET-SALES-DB
    .
    .
    .
GET CUSTOMER KEY = 1236204
```

In this case, the GET command will retrieve the CUSTOMER record using the specified storage-key value. Note that the CUSTOMER record was defined to the Dictionary with the CUSTOMER-NUMBER field specified in the STORAGE-KEY option.

Example: Obtaining a Buffer or Records

```
CONTROLLED FILE DEFINITION FOR WIDGET-SALES-DB
PATH DEFINITION
WHERE: BUFFER (CUSTOMER, ORDER OCCURS 5)
   .
   .
GET WIDGET-SALES-DB KEY = 1236204
```

In this case, the same CUSTOMER record would be retrieved, but it would be accompanied by the first five ORDER records associated with that CUSTOMER.

Example: Random Access Using Index Value

```
CONTROLLED FILE DEFINITION FOR WIDGET-SALES-DB
PATH DEFINITION
WHERE: BUFFER (CUSTOMER VIA CUST-NAME-IX)
   .
   .
GET CUSTOMER KEY = 'BREVIK MINI INC.'
```

In this case, the CUSTOMER record would be retrieved using the Name index. Note that the index CUST-NAME-IX was defined to the Dictionary with the CUSTOMER-NAME field specified in the BASED ON option.