Britton Lee Host Software

IDMFCOPY USER'S GUIDE

(R3v5m2)

March 1988

Part Number 205-1680-001

Table of Contents

Preface

This manual is an introduction to the use of the utility *idmfcopy*. **It should** be read in conjunction with the reference material pertaining to *idmfcopy* in **Section (1I)** of the *Host Software Specification* (UNIX Systems) or in the *Command Summary* (other systems).

There are a number of examples throughout this guide which illustrate various invocations of *idmfcopy*. In all cases, both a UNIX version of the command-line precedes a non-UNIX version, as illustrated below. In cases where the command-line requirements for VM/CMS hosts are more restrictive, a third VM/CMS example is given.

```
idmfcopy in –f "emp.fmt"

idmfcopy in /formatfile="emp.fmt"

idmfcopy in mydatabase emps /formatfile="emp.fmt"
```

One-character options preceded by a hyphen, such as **–f**, are used in UNIX environments. One-word options preceded by a slash, such as **/formatfile**, are commonly used in other host environments. In this example, **–f** and **/formatfile** represent the same option in UNIX and non-UNIX environments.

We have used the terminology of Britton Lee's Intelligent Database Language (IDL) throughout this guide to refer to certain database objects and commands. The table below can be used to translate these IDL terms to the Structured Query Language (SQL).

IDL	*SQL*
relation	table
tuple	row
attribute	column
append	insert
retrieve	select

Keywords and command-lines which can be entered at the terminal **are shown** in boldface.

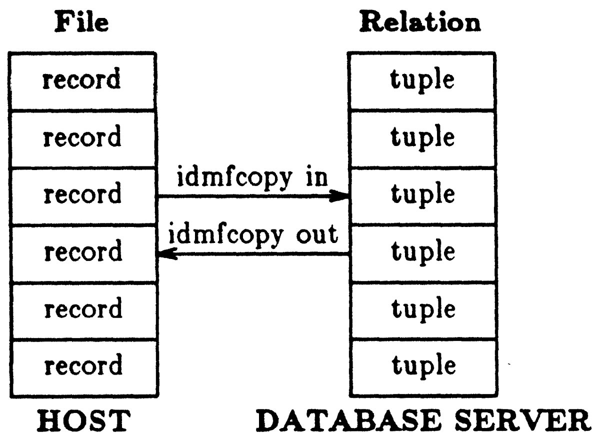
1. General Information

1.1. Definition

Idmfcopy converts and copies formatted data between a file on the **host system** and a relation on the Britton Lee database server.

Idmfcopy consists of two programs. *Idmfcopy in* reads records from a file or tape on the host computer, converts them to a format which can be read by the database server, and copies them into a relation on the database server. *Idmfcopy out* reads tuples from a relation on the database server, converts them to a format specified by the user, and writes them as records to a file on the host computer.

A record is converted and copied to a tuple, or vice-versa, one at a time.



1.2. Uses

One common use of *idmfcopy in* is to load a new relation into the database server from an existing host file. *Idmfcopy in* can also be used to append a large amount of data to an existing relation. To add small amounts of data, the IDL command **append** is appropriate, but it is more efficient to use *idmfcopy in* to add large amounts of data to a relation.

If it is necessary to edit a large amount of data in a relation, it is often easier to copy the data out and edit it on the host system using a text editor rather than update the relation using the database query languages IDL or SQL. The edited host file can then be copied back in, after you have deleted the tuples in the original relation.

Idmfcopy is also an appropriate tool for moving data from one host to another via a database server. Relations can be copied in from one host to the database server and then out from the database server to another host.

1.3. Other Copy Utilities

Idmfcopy is distinguished from other Britton Lee copy utilities in that it copies user-formatted data. If you are working with data formatted by the database server, it would be more appropriate to use other utilities to copy your data. The utility *idmcopy* is used for copying database-server-formatted data to and from the database server. If you wish to back up an entire database, use *idmdump* and *idmload*. These utilities are described in Section (1I) of the *Host Software Specification* (UNIX Systems) or in the *Command Summary* (other systems).

1.4. Preparation

You will have to provide certain information to *idmfcopy*. From the database server side, you must provide the name of the database and the names of the the relations you are copying to or from, and the names of the attributes in those relations. From the host side, you must supply the name and type of the host file being copied to or from and some information about how data is organized in the file.

For both *idmfcopy in* and *idmfcopy out*, the relation must already exist. For *idmfcopy out*, the host file will be created if it does not already exist.

You also need to create a format specification which *idmfcopy* uses to convert data. This specification is based upon information you have about the host file and the relation. Learning to use *idmfcopy* primarily involves learning to build this format specification.

For the examples used in this guide, we will create the host file and the relation so that you can see how this material is generated. In most real-life applications, however, the host file and relation already exist. You will have to examine the file, and any available documents describing it, especially if it contains binary data, in order to obtain the information necessary to build the format specification. You also need to obtain some specific information about the relation; at a minimum you need to know the names of the attributes which will be affected by *idmfcopy*, since these names must be used in the format specification. It would also be useful to know the types and sizes of the attributes in order to understand any error messages generated by *idmfcopy*.

2. A Simple Example

2.1. The Host File

Let us create a host file named "empfile" containing the employee numbers, last names, first initials, and phone extensions of the employees in an organization. This can be done with the text editor that we normally use on our host:

```
101Claus,N20
102Hood,R21
103Kole,K26
104White,S28
```

All the records in this file have the following structure:

> The first field, containing the employee number, consists of three digits.
>
> The second field, containing the last name, consists of a variable number of alphabetic characters and is terminated with a comma.
>
> The third field, containing the first initial, consists of one alphabetic character.
>
> The fourth field, containing the telephone extension, consists of two digits.

This choice of organization is completely arbitrary. Later in this document, we will demonstrate other methods of describing and delimiting fields containing the same and similar data.

2.2. The Relation

Let us create a relation "emps" on the database server, whose attributes correspond to the fields in "empfile". We can do this using the *idl* program

```
1) open mydatabase
2) create emps (empno = i2, lname = c12, init = c1, ext = i1);
1) exit;
```

or the *sql* program

```
1) open mydatabase -
2) create table emps (empno smallint, lname char(12), init char(1), -
3) ext tinyint)
1) exit
```

Either of these commands creates an empty relation in "mydatabase" which looks like this:

empno	lname	init	ext

The names of the attributes (empno, lname, init, ext) are the same names which will be used in the format specification when *idmfcopy* is invoked to copy to or from this relation.

2.3. The Format Specification

Whether you wish to copy in or out, you need to build a format specification to describe the host file and the relation to the *idmfcopy* program. The simplest way to do this is to create a format specification file or *formfile* using the text editor on your host system.

The following format specification file named "emp.fmt" can be used to copy data between the host file "empfile" and the relation "emps":

```
database    mydatabase;
relation    emps;
file        "empfile";
record
        empno decimal(3);
        lname text to comma;
        init  text(1);
        ext   decimal(2);
end
```

This format specification maps the attributes in the "emps" relation, represented by the attribute names (empno, lname, init, ext), to fields in the host file "empfile". The fields are described by their type and method of delimitation (decimal(3), text to comma, etc.). This descriptive portion of the format specification always describes the data as it appears in the host file, not the relation, no matter which direction *idmfcopy* is copying. The corresponding attribute is represented by its name alone. Therefore, the field corresponding to the attribute "lname" is described as "text to comma", which describes its structure in the host file, not as "text(12)" which would reflect the allocation of 12 characters for the attribute "lname" in the relation.

If there are incompatibilities between the way in which a field appears in the host file and the description of its corresponding attribute in the relation, *idmfcopy* will treat them in a predictable manner. In this case, if the value of a last name occupies fewer than 12 characters, the value in the relation will be blank-padded. If it occupies more than 12 characters, a warning will be generated and the record containing the illegal field will not be copied. We will describe a method for previewing such problems in the section on error checking.

2.4. Idmfcopy In

Thus far, we have created a host file called "empfile", a relation called "emps", and a host file called "emp.fmt" containing a format specification.

The relation "emps" is empty. We can now invoke *idmfcopy in* to load it with the data from "empfile":

```
idmfcopy in -f "emp.fmt"

idmfcopy in /formatfile="emp.fmt"

idmfcopy in mydatabase emps /formatfile="emp.fmt"[1]
```

The **-f** or **/formatfile** option instructs *idmfcopy* to read the *formfile* called "emp.fmt" for the name of the host file, the name of the relation and the attributes in the relation along with a description of the fields in the host file which correspond to those attributes.

If you wish to verify that *idmfcopy in* was successful, retrieve all of the attributes in the relation using IDL

```
1) range of e is emps;
1) retrieve (e.all);
```

or SQL

[1]If the host environment is VM/CMS, the database name and the relation name must be specified on the command-line when *idmfcopy* is invoked.

```
1) select * from emps
```

empno	lname	init	ext
101	Claus	N	20
102	Hood	R	21
103	Kole	K	24
104	White	S	25

If the *idmfcopy in* did not produce the desired results, all of the data in the relation can be deleted with the **truncate** command, which is identical in IDL and SQL:

```
1) truncate emps
```

2.5. Idmfcopy Out

If *idmfcopy in* was successful, the relation can be copied out, often using the same format specification that was used to copy in. There are some cases in which it is not advisable to use the same format specification to copy in and out. These cases are discussed under "Delimited Text Fields".

If you wish to copy out to a new host file, you can override the filename in the format specification by specifying another filename on the command-line. The new filename can be specified with the **–d** or **/datafile** option:

```
idmfcopy out –f "emp.fmt" –d "newempfile"

idmfcopy out /formatfile="emp.fmt" /datafile="newempfile"

idmfcopy out mydatabase emps /formatfile="emp.fmt"
/datafile="newempfile"
```

Since no modifications were made to the relation, "newempfile" should be identical to "empfile". In general, if one filename is indicated on the command-line and another in the format specification, the filename on the command-line takes precedence.

3. Building a Format Specification

At a minimum, a format specification consists of a series of statements describing the data in the host file and a mapping of host file data to attributes in the relation. A format specification is needed whether data is being copied in or out of the database server.

Other format specification statements, such as those indicating the database and relation or the name of the host file, may be provided as parameters on the command-line when *idmfcopy* is invoked or included in the *formfile* as in the previous example.

We could have invoked *idmfcopy* by

```
idmfcopy in -f "emp.fmt" -d "empfile" mydatabase  emps

idmfcopy in mydatabase emps /formatfile="emp.fmt"
/datafile="empfile"
```

Then the entire format specification would have looked like this:

```
record
        empno decimal(3);
        lname text to comma;
        init  text(1);
        ext   decimal(2);
end
```

This example has the exact same meaning as the one in Section 2.3. In that example, the names of the host file, the database, and the relation were included in the format specification itself, while here they are indicated on the command-line.

If the database name, relation name, or host file name is indicated both on the command-line and in the format specification, the name on the command-line takes precedence.

If your host environment is VM/CMS, the alternative of supplying the database name and relation name in the format specification is not available. If the database name or relation name is included in the format specification, is ignored. In this environment, the only correct invocation is

```
idmfcopy in mydatabase emps /formatfile="emp.fmt"
/datafile="empfile"
```

In order to write the part of the format specification which actually describes the organization of the data in the record, it is necessary to have specific information about the *file* being copied to or from, the *records* in that file, and the *fields* contained in the records.

3.1. Files

3.1.1. File Structures

Host files are considered to have a stream-based or record-based presentation, depending upon the way in which their data is conceptually organized. A file with a stream-based presentation is perceived as a stream of bytes with no notion of record boundaries imposed on it. Data is read from or written to a stream-based file by specifying a starting point and the number of bytes to be read or written.

A file with a record-based presentation is perceived as discreet chunks of data called records, in which each chunk has a clear beginning and end. Records may be of variable or fixed length. The record boundaries are created when the file is written, and reads performed on the file must pair one-to-one with the writes. Data is read from or written to a record-based file by specifying the records to be read or written.

The conceptual presentation of a file is not necessarily the same as its actual physical structure, so that it is possible to mix physical structures and presentations. Generally, a record-based presentation may be used on physically record-based files or on physically stream-based files. A stream-based presentation may be used on physically stream-based files; however, an attempt to perform stream-based reads and writes on a physically record-based file may produce undefined results.

It is helpful to know both the physical structure and the presentation of the host file before attempting to build a format specification. The possible physical structures and presentations for host files in environments currently supported by Britton Lee Host Software are:

ENVIRONMENT	*PHYSICAL STRUCTURE*	*PRESENTATION*
UNIX	stream	stream or record
VMS	stream or record	stream or record
VM/CMS	record	stream or record
PC MS-DOS	stream	stream or record

3.1.2. File Specifications

Host data may reside in a disk file or tape file or it may be entered from the user's terminal. General information about a host file is detailed in a ***filespec***. The ***filespec*** can be a quoted string in the format specification; in the example in Section **2.3**, the word "empfile" is the *filespec*. The *filespec* can also be a parameter on the command-line; in the example on page 7, the "empfile" following the **–d** or **/datafile** option on the command-line is the *filespec*.

Filespecs can include optional parameters, some of which are discussed below. For the format of a *filespec*, consult the entries for *ifscrack()* and *ifopen()* in the *Host Software Specification* or *C Run-Time Library Reference*.

3.1.2.1. File Specification Parameters

Optional parameters may be included in a *filespec* to override default assumptions which *idmfcopy* makes about a file. The parameters are separated from the filename by a filename delimiter, which is usually a percent symbol (%), except on VMS hosts where it is a score (#).

If the file resides on host tape, this can be indicated in the *filespec* as

```
empfile%htape
```

or

```
empfile#htape
```

if the host is VMS. If the file type is not specified, the default is a disk file.

In addition to the file type, there are other parameters which can be used to override default assumptions about a host file. Some useful parameters are listed below:

mode(m)	If *m* is **a**, the file is open for appending. The default on copy out is **w** signifying open for writing. When *m* is **w**, any data already in the host file is over written on copy out.
rbp(b)	If *b* is 0, record-based presentation is off. If *b* is 1, it is on. This parameter is specified when the host file has a physical record-based structure and all of the fields are binary and the record structure is not specified elsewhere in the format specification.
rs(n)	This indicates the record size. On VMS, if the record size is not specified, the block size **bs** is used. If record size is not specified in the format specification, the system provides **a default** record size on physically record-based systems. On physically stream-based systems, if the fields in a record are all fixed-length, the

default record size is the sum of the fixed-length fields. If the fields are of variable length, the default record size is 8 * IOBSIZE.

bs(n)	On output, this defines the physical block size. On input, the file type can override this value if the block size can be determined from the file itself.
format(f)	For ANSI-labeled tape, **f** indicates fixed format, **d** indicates variable format. The default is **d**. This feature is not supported on all systems.
padchar(c)	This specifies the character to use for padding on files having a stream-based physical structure. It defaults to binary zero on most file types.
type(c)	If *c* is **b**, the file is opened as binary; if *c* is **t** it is opened as text. The default is text.

For example,

```
empfile%hfile,mode(a),rbp(1),rs(16)
```

specifies that the data is in a host file to be opened in append mode, using a record-based presentation and a record size of 16 bytes, and

```
empfile%htape,format(f),padchar($)
```

indicates that the data is to be read from a host ANSI-labeled tape in fixed format and that unfilled portions of fields are to be padded with the "$" character on *idmfcopy out.*

3.1.2.2. Default Host Files

If there is no *filespec* in the format specification or on the command-line, the host file is assumed to be the user's terminal (or *stdin* and *stdout* on a UNIX system). For example, the command

idmfcopy out mydatabase emps –f "emp.fmt"

idmfcopy out mydatabase emps /formatfile="emp.fmt"

when "emp.fmt" looks like

```
record
   empno       decimal(3);
   lname       text to comma;
   init        text(1);
   ext         decimal(2);
end
```

has no *filespec*. It produces the following output at the user's terminal.

```
101Claus,N20
102Hood,R21
103Kole,K26
104White,S28
```

3.2. Records

Files consist of one or more records, and records consist of one or more fields. We categorize records according to the way in which their boundaries are delineated. There are four record types of interest to *idmfcopy*:

- fixed-length
- delimited
- natural
- field-delimited

A single host file will contain only one type of record. The type of record which exists in a particular file depends on the characteristics of the data and the physical structure and presentation of the host file.

3.2.1. Fixed-Length Records

Fixed-length records are typical for a host file with a record-based presentation and fixed-length fields.

The format for specifying a fixed-length record of 80 bytes is

record (80).

On an *idmfcopy in* of a fixed-length record, if the specified record size *n* is less than the actual record size in a physically record-based file, data beyond the *n*th byte will not be copied. On an *idmfcopy out,* if the data in the tuple is less than *n* bytes, the record will be padded to fill the file's specified record size.

3.2.2. Delimited Records

The end of a delimited record is marked with a special delimiter character. A list of commonly used delimiters which may be referenced by their symbolic names is provided in Appendix A.

A single host file can contain delimited records of varying lengths. On copy out, the record is scanned until the specified delimiter is found, and then it is copied. More than one character can be specified as the delimiter. In this case, on an *idmfcopy out* the first delimiter specified is written to the host file. On an *idmfcopy in*, the first delimiter encountered is interpreted as the end of the record.

It is essential that the delimiter character not appear in any part of the record except at the end. The delimiter itself is not considered part of the record and is discarded when the data is copied.

A record which is delimited by a newline character is represented in the format specification as

```
record to nl
```

The delimiter may be specified by its symbolic name, its graphic symbol, or its ASCII or EBCDIC numeric value (hex or octal), so that

```
record to nl
record to '\n'
record to 012          /* ASCII */
record to 15           /* EBCDIC */
```

all signify that the record delimiter for a specified host file is a newline.

3.2.3. Natural Records

A natural record is specified as

```
record
```

This is the simplest record specification format to use, because it leaves to *idmfcopy* the task of interpreting the size and method of delimiting the record.

When the user specifies a natural record, *idmfcopy* determines the presentation and physical structure of the host file and extrapolates from the format specification certain information about the fields. *Idmfcopy* then interprets the structure of a natural record according to the following guidelines:

(1) If all the fields in the record are of fixed length, the file is assumed to have a record-based presentation. The record is treated as a fixed-length record with its length equal to the sum of the lengths of the fields.

(2) If the file has a record-based physical structure, a record-based presentation is assumed. If all the fields are fixed length, the record is treated as in (1) above. If not all the fields are of fixed length, a system default length is used for the length of the record.

(3) If the file has a stream-based physical structure and not all fields in the record are of fixed length, a stream-based presentation is assumed. The record is treated as a delimited record with the newline (nl) as its delimiter.

Our simple example in Section 2.3 uses a natural record specification. If the host file "empfile" had a record-based physical structure the file would be assumed to have a record-based presentation (guideline 2). Since the field length for the field containing the last name is not fixed, a system-provided default length would be used for the length of the record.

If "empfile" had a stream-based physical structure, a stream-based presentation would be assumed (guideline 3), and the record would be considered to be of variable length delimited by a newline.[2]

3.2.4. Field-Delimited Records

Field-delimited records are copied in or out a field at a time. The format for specifying a field-delimited record is

record (*)

Field-delimited records will not work on a file with a record-based physical structure and are not recommended for files with stream-based physical structures either, as they afford less reliable error detection and recovery than the three record specifications previously discussed. This feature is provided to allow unusual data sets to be copied in.

[2] If the host environment is VM/CMS, a record in which all the data is of type text must be specified as **record to nl** and a record containing any binary data must be specified as **record**.

3.3. Fields

A record consists of one or more fields. Just as a record in a host file corresponds to a tuple in a relation on the database server, a field in a record corresponds to an attribute in a tuple. Each attribute has a unique name which is used to pair a field description with its corresponding attribute in the format specification.

RECORD

101Claus,N20

TUPLE

empno	lname	init	ext
101	Claus	N	20

FORMAT SPECIFICATION

```
record
        empno decimal(3);
        lname  text to comma;
        init   text(1);
        ext    decimal(2);
end
```

There are two reserved attribute names: **all** for specifying all the fields in a record or attributes in a relation and "-" for specifying a dummy field.

If the name of an attribute is the same as that of an *idmfcopy* keyword, it must be enclosed in quotation marks in the format specification. For example, if an attribute name is identical with the keyword **text**, it must be specified as **"text"** in the format specification. Keywords are listed in Appendix B.

Fields are described in the format specification by a field specification or *fieldspec*. This section contains informal descriptions and examples of field specifications for various types of data. For a formal description of the syntax of a *fieldspec* and all of its components, consult the reference material for *idmfcopy* in the *Host Software Specification* and *Command Summary*.

All field specifications begin with the name of the attribute which corresponds to the field in the host file. The remainder of the field specification describes the structure of the field in the host file.

Fields are specified according to the types of data they contain (text or binary). Data stored in human-readable form is text data, data stored in machine-readable form is binary data. Within this classification based on the type of the data, fields may be described further by the manner in which their boundaries are delineated. At this level, a field may be considered fixed-length, delimited, or counted.

3.3.1. Text Fields

There are six text types: **text** (for character strings), and **decimal**, **hex**, **octal**, **sci**, and **float** (for numbers). On *idmfcopy in*, **sci** and **float** are identical; on *idmfcopy out*, **sci** formats data in exponential notation.

3.3.1.1. Fixed-Length Text Fields

The description of a fixed-length text field consists of the name of the corresponding attribute, the text type, and the length of the field (in digits or characters) enclosed in parentheses. If the field is of type **sci** or **float**, an optional precision may be indicated by a comma followed by an integer to represent the number of digits to the right of the decimal point. If the precision is not specified, a default precision of 6 digits is used on *idmfcopy out*.

Below are some examples of field specifications for fixed-length text data:

```
lname  text(12)
empno  decimal(3)
salary float(4,2)
```

When fixed-length fields do not have enough data to fill their specified lengths, numeric data is blank-padded on the left and text data is blank-padded on the right on *idmfcopy out*.

Numeric data may be preceded by a "+" or "–" sign. A field containing hexadecimal or octal data may contain the "0x" or "0o" base notations. The characters used to indicate sign and base notations are not counted as part of the length of a field when the length of a numeric field is specified.

3.3.1.2. Delimited Text Fields

If the length of a field is not fixed, the end of a field may be marked with a delimiter character.[3] The delimiter character for a specific field must not appear as data in that

[3]If the host environment is VMS and the file is stream-based, the line-feed, form-feed, and carriage-return characters should not be used as field delimiters, because they may be recognized by VMS as record delimiters. If the host environment is PC/MS-DOS, and a field delimiter is not ASCII text, tab, or newline, the file should be opened as binary. This

field, nor should a field delimiter be identical with the record delimiter for the record which contains the field.

Delimited fields are specified by the name of the corresponding attribute, the text type, the keyword **to** and the field delimiter. The delimiter may be specified by its symbolic name, its graphic symbol, or its numeric value:

```
lname text to ','
lname text to 054        /* ASCII */
lname text to 6B         /* EBCDIC */
lname text to comma
```

We could have built the records in our sample "empfile" completely with delimited fields. This would have been necessary had we wanted to accommodate variable-length employee numbers, initials, and phone extensions as well as variable-length last names. The file might have looked like

```
101 Claus,N:20
102 Hood,R:21
103 Kole,K:26
104 White,S:28
```

in which case the format specification would have looked like

```
record
        empno decimal to space;
        lname text to comma;
        init  text to colon;
        ext   decimal to nl;
end
```

You can execute *idmfcopy in* substituting this new file and format specification for the original ones in Section 2.3 and produce the exact same tuples in the relation.

It is important to understand that, although the lengths of delimited text fields are not fixed in the host file, a fixed amount of storage was allocated for each attribute when the relation was created on the database server. Field lengths which exceed the storage allocated for their corresponding attributes will generate overflow warnings and not be copied in.

can be done by indicating type(b) in the *filespec*. See Section 3.1.2.1.

3.3.1.2.1. Default Delimiters

A set of default field delimiters may be specified with the delimiter statement in the format specification. For example, the semicolon can be specified as a default delimiter as follows:

```
delimiters    ;
```

If the delimiter statement is not used and no delimiter is named in the field specification, tab, comma, and newline are the default delimiters on copy in, and comma is the only default delimiter on copy out. The tuple

empno	**lname**	**init**	**ext**
101	Claus	N	20

could be copied out using the format specification

```
record to nl
        empno text;
        lname text;
        init  text;
        ext   text;
end
```

In this case, default field delimiters would be inserted to produce a record that might look like this:

```
101,Claus,N,20,
```

To copy the tuple back in, the same format specification could be used, but there is always a danger that one of the other default delimiter characters could exist in the data. If this were the case, the character could be interpreted as a field delimiter by *idmfcopy*. Even if default delimiters are being used to copy data out, it is more prudent to state field delimiters explicitly when copying in. The safest format specification to copy in this record would be

```
record
        empno text to comma;
        lname text to comma;
        init  text to comma;
        ext   text to comma;
end
```

3.3.1.3. Counted Text Fields

Another way of specifying variable-length fields is by using counted fields. In a counted field, the data is preceded by a one-byte count field which contains the number of bytes of data in the field. This count does not include the count field itself.

For example, a counted field consisting of eight bytes of data can be conceptualized as

count	**data**
8	eight bytes of data

If the fields in the file have been set up as counted fields, you must specify them as such, unless you are prepared to make extensive revisions to the host file to accomodate another field structure.

A counted field is specified by the corresponding attribute name and the field type followed by an asterisk enclosed in parentheses.

```
lname  text(*)
empno decimal(*)
```

A counted field may also be specified by the attribute name and the field type followed by the keyword **var**.[4] When this form is used, the count field occupies two bytes:

```
lname  text(var).
```

[4]The **var** form is commonly used on IBM systems.

3.3.2. Binary Fields

There may be fields in your host file, or even entire records, which contain binary rather than text data. There are two categories of data which may be stored in binary format: numeric data which is stored in fixed binary format, and decimal numbers which are "packed" or binary coded (**bin**, **bcd**, **bcdflt**).

Data may be stored in binary format because it requires less storage in this form or because manipulation of numeric data is more efficient since binary data need not be converted to machine-readable form prior to being processed. On the other hand, *data specified in binary format is not readily convertible from one host machine to another. Therefore, binary specifications are not recommended when the file may be transferred to a host supporting a different internal representation.* If there is any possibility that a host file will be ported to another system, format specifications should utilize a text format, which will automatically be converted to the appropriate internal representation by *idmfcopy*.

3.3.2.1. Fixed-Length Binary Fields

A fixed-length binary field specification consists of the corresponding attribute name followed by a *fixedbinspec* which indicates both the type of numeric data (integer or floating-point) and the number of bytes of data in the binary field. Possible *fixedbinspecs* are

i1	(1-byte integer)
i2	(2-byte integer)
i4	(4-byte integer)
f4	(4-byte floating-point number)
f8	(8-byte floating-point number)

The following are typical fixed-length binary specifications:

```
salary f4
empno i2
```

The binary-coded decimal data types (**bin**, **bcd**, **bcdflt**) may have fixed-length specifications consisting of an attribute name, followed by one of these **bcd** types, followed by the length of the field in bytes enclosed in parentheses. The precision of a **bdcflt** is indicated by a comma followed by an integer indicating the precision:

```
salary bcdflt(4,2)
empno bcd(2)
```

3.3.2.2. Counted Binary Fields

Binary-coded decimal fields which do not have a fixed length are specified as counted fields. As with counted text fields, the first byte is the length byte which indicates the number of bytes in the field. Counted **bcd** fields are specified by the corresponding attribute name followed by the **bcd** type followed by an asterisk enclosed in parentheses:

```
num    bcd(*).
```

3.3.3. Combining Field Types

Text and binary fields can be combined in the same record. For example, a record with the format specification

```
record
        empno decimal(3);
        lname text(12);
        init  text(1);
        ext   i2;
        salary bcdflt(4);
end
```

is treated as a fixed-length record of 22 bytes. The host file is opened as a binary file, because at least one of the fields is binary.

Fields which are delimited in different manners can be combined within the same record. A file described by the format specification

```
record
        empno decimal(3);
        lname text to comma;
        init  text to comma;
        ext   i2;
        salary bcdflt(4);
end
```

would be treated as a file of records delimited by newlines if the host file had a stream-based physical structure. If the file had a record-based physical structure, it would be considered a file of fixed-length records with a default record length provided by the system.

3.3.4. Dummy Fields

It may be desirable to omit some of the fields when the record or tuple is copied. This is done by specifying the field to be omitted as a dummy field. Dummy fields are signified by a hyphen:

```
record
        -       decimal(3);
        lname   text to comma;
        init    text to comma;
        ext     i2;
        -       bcdflt(4);
end
```

The first and fifth fields from the host file are discarded on the *idmfcopy in*.

Dummy fields are not copied in to the database server, but they are copied out to the host file where they are given a null value appropriate to their type.

3.3.5. Initialized Fields

Initializing fields is a common practice when copying out dummy fields, so that the dummy fields can be easily identified in the host file. For example,

```
record
        empno   decimal(3);
        -       text(12) = "############";
        init    text(1);
        ext     i2;
        -       bcdflt(4) = 00.00;
end
```

produces a record on *idmfcopy out* which might look like

```
103############K2600.00
```

When an initialized field is copied in, the initialization specified in the format specification must match the actual data in the host file.

4. Error Checking

On an *idmfcopy in*, records which do not fit the format specification generate a warning or error message and are not copied into the relation. Warnings are usually caused by conversion overflow. Records which generate warnings can be copied into the relation if *idmfcopy* is invoked with the **–w** or **/warning** option, which instructs *idmfcopy* to ignore warnings.

Records which generate warnings and errors may be collected in a *rejectfile* which is created automatically on the host. To create a *rejectfile*, invoke *idmfcopy in* with the **–r** or **/rejectfile** option followed by the name of the *rejectfile*. A common use of this feature is to combine it with the **–n** or **/checkdata** option. This performs all the conversions without copying the records, so that the records in the *rejectfile* can be examined, edited in the host file, and resubmitted.

The command

```
idmfcopy in mydatabase emps –f "emp.fmt"  –r "badrecs"

idmfcopy in mydatabase emps /formatfile="emp.fmt"
/rejectfile="badrecs"
```

copies records into a relation, placing all the records that do not fit the specification into a host file named "badrecs". The command

```
idmfcopy in  mydatabase emps –f "emp.fmt"  –n –r "badrecs"

idmfcopy in mydatabase emps /formatfile=emp.fmt
/checkdata /rejectfile="badrecs"
```

formats the data in preparation for copying, copies the bad records into "badrecs", but does not actually perform the copy to the database server. The user may then examine the bad records, edit the host file or the format specification, and then resubmit the file for copying. Rejectfile processing is not available with field-delimited records.

Occasionally there may be so many erroneous records in a host file that it would be advisable to halt the *idmfcopy* process. Invoking *idmfcopy* with the **–e** or **/errorstop** option followed by an integer instructs the program to stop processing after the specified number of errors.

If you to wish see the data which has been copied displayed at the terminal, invoke *idmfcopy* with the **–v** or **/verbose** option.

The table below gives a summary of the error-checking options offered by *idmfcopy*:

OPTION (UNIX)	*OPTION (OTHER)*	*MEANING*
–e *n*	**/errorstop=***n*	Stop processing after *n* errors.
–n	**/checkdata**	Check data, but do not copy.
–r *rejectfile*	**/rejectfile=***rejectfile*	Collect errors in *rejectfile.*
–v	**/verbose**	Use verbose mode: display data being transferred.
–w	**/warning**	Ignore warnings.

5. Batches

By default, *idmfcopy in* copies 5000 records together. As each 5000-record batch is committed, a message is printed informing the user of the number of records copied so far.

This feature is useful for copying large amounts of data. If the system crashes, the database server will back out any portions of the batch which have not been committed. Upon recovery, you can skip the records which have already been committed using the **–s** or **/skipdata** option with an integer to indicate the number of records to skip:

```
idmfcopy in mydatabase emps –f "emp.fmt" –s 5000

idmfcopy in mydatabase emps /formatfile="emp.fmt"
/skiprecord=5000
```

The 5000-record default batch size can be overridden by invoking *idmfcopy* with the **–b** or **/batchsize** option with an integer to represent the desired batch size in number of records:

```
idmfcopy in mydatabase emps –f "emp.fmt" –b 2000

idmfcopy in mydatabase emps /formatfile="emp.fmt"
/batchsize=2000
```

Appendix A: Symbolic Delimiters

This is a list of delimiter characters which may be used to delimit **records and** fields. These delimiters may be referenced by their symbolic names (i.e. **null**, **comma**) in a format specification.

Name	*Graphic*	*ASCII*	*EBCDIC*	*Meaning*
null		000	00	Null
tab	\t	011	05	Horizontal Tab
nl	\n	012	15	Newline
lf		012	25	Line Feed
ff	\f	014	0C	Form Feed/New Page
cr	\r	015	0D	Carriage Return
fs		034	22	Field Separator
gs		035		Group Separator
rs		036	35	Record Separator
us		037		Unit Separator
space	( )	040	40	Space
comma	,	054	6B	Comma
dash	–	055	60	Dash/Hyphen/Minus
dot	.	056	4B	Dot/Period/Decimal Point
slash	/	057	61	Slash
colon	:	072	7A	Colon
semi	;	073	5E	Semicolon

Appendix B: Idmfcopy Keywords

The following words have special significance to the *idmfcopy* program and must be enclosed in quotation marks if they are used as the names of attributes in a relation on a database server.

all	**f8**	**relation**
bcd	**file**	**sci**
bcdflt	**float**	**text**
bin	**hex**	**to**
database	**i1**	**unsigned**
delimiters	**i2**	**var**
decimal	**i4**	**verbose**
end	**octal**	
f4	**record**	