RMS

Record Management System
for the OS-9 operating system

RMS: Record Management System for the OS-9 operating system

Copyright © 1983 Dragon Data Ltd

RMS, in all machine readable formats and the written documentation accompanying them, are copyrighted. The purchase of RMS conveys to the purchase a licence to use RMS for his/her own use and not for sale or free distribution to others. No other licence, expressed or implied, is granted.

This manual and associated programs are the copyright of Dragon Data Ltd (under licence). Copying is only permitted for backup purposes on a single computer system. Any other copying by whatever means without the written authorisation of Dragon Data Ltd is strictly prohibited.

This manual was prepared using the stylograph word processor, check with the spell check spelling checker, and printed with the mail merge merge/print program.

OS-9 and BASIC09 are trade marks of Microware Systems Corporation and Motorola Inc.

System Requirements

RMS is supplied configured for your Dragon Data Computer, and will work equally well under OS-9 Level One or OS-9 Level Two. The programs do not require large amounts of working memory; each takes 8k bytes or less, plus the size of the program.

The minimum system configuration you require is your Dragon Data compute, with at least 64k bytes of RAM, running the OS-9 operating system; a floppy disk drive; and a television, or preferably a monochrome monitor. (The monitor will generally give you a better picture than a television).

RMS will certainly run with only one disk drive, although two are more convenient.

A printer is not essential with RMS, but it does increase the usefulness of RMS. Almost any printer can be used - RMS makes no assumptions about the printer, not does it use any special printer features.

You will need a text editor - such as the stylograph word processor, or the OS-9 text editor - to create the RMS dictionaries and report specifications.

Chapter 1. Introduction to RMS

Record Management System is a complete DATABASE MANAGEMENT SYSTEM for OS-9. It is extremely versatile, and very easy to learn to use. Applications may include accounting, business record-keeping, management information systems, customer or personnel records, customized data entry, immediate data retrieval, and various other situations which require data entry, online data retrieval and update, and printed reports. RMS can easily be customized to fit various requirements without any programming knowledge. The data stored under RMS is, however accessible to user written programs (in BASIC09 or other languages).

1.1. What RMS can do

RMS allows the user to determine the format under which his data is to be stored. This format consists of deciding what data items are to be kept, and what the characteristics of each item are to be. The characteristics include the type of data (ie, number, alphanumeric string, date, money value), the maximum length of the data item, and if necessary, limits or restrictions on the possible values.

Once the data format is determined and stored in a DICTIONARY, then RMS automatically provides:

- creation of a disk file properly formatted to store the data,
- online data entry in a form fill-out manner on the screen,
- online data access for lookup or modification,
- creation of printed reports and text files to the user's specifications,
- facility to later change the format of the data as requirements change.

1.2. What RMS consists of

RMS is a package of utility programs, written in machine language for maximum efficiency. These utilities may be used individually or in various combinations to provide the above features. It is also possible to access RMS data file from user-written programs, for example in BASIC09. This allows the user to take advantage of RMS to do much of the time-consuming part of a large custom programming task, and still allows for the use of a general purpose programming language for the unique processing that can be done no other way. The RMS programs are:

RMSNEW	is used to create and format a new RMS data file which fits the user's needs.
RMS (EDITOR)	is used to input data to a file, modify data in a file, or display data in a file. In general, this is accomplished by the display of a form on the screen. The form reflects the specific data items as determined by the user. The user can fill out the form to enter new data or can request data from the file to be displayed in the form.
REPORT	is used to create printed or text file output from the RMS data file. The user has complete control over the data that is printed and the manner in which it is printed. Many built-in features are available to facilitate the creation of quality finished reports, or text files for use by other programs, such as MAIL MERGE.
INDEX	is used to create one or more index files from the data file which can be used to "drive" the RMS EDITOR or REPORT programs. This provides display, modification or printing of the data in the desired order.
RMSCOPY	is used to copy an RMS file when its capacity or internal structure is required to be changed. It can also be used to merge two files or to "post" one file to another.

1.3. Getting Started

Important

We suggest that you read this short section carefully, and then try out the example application at the end of the manual. The assignment of the command keys for RMS on your keyboard and a summary of OS-9 command lines for calling RMS are given at the back of this manual.

The first thing you must do before using RMS is to make a copy of the supplied RMS onto a freshly formatted disk. Then *store the original disk in a safe place*. *Never* use the supplied disk for running RMS, and *never* write to it. Refer to the *OS-9 User's manual* for instructions on how to use the **format** and **backup** commands.

Remove your system disk from drive 0 and insert your freshly made disk. Change your execution directory (using 'chx') and your data directory (using 'chd') to refer to this disk:

```
OS9: chx /d0/cmds chd /d0
```

If you are using a Dragon 64, and the screen is not already in the 51 characters by 24 lines mode, enter this mode by:

```
OS9: go51
```

Inspect the root directory of the disk by:

```
OS9: dir
```

The root directory contains several example files:

SAMPLE.DIC	example files used in the tutorial
PRLIST.REP	at the end of this manual
ADDRESS.RMS	a sample data file - an address book
ADDRESS.DIC	the dictionary file for the address book
BOOK.REP	a report spec to print out the address book
PHONES.REP	a report spec to print out the phone numbers
NAME.NDX	a sample index file for the address book

In addition there are two directories, CMDS and SYS. CMDS contains the RMS programs. To inspect the directory, try:

```
OS9: dir cmds
```

RMS	the RMS editor, to enter or modify records
RMSNEW	for creating new data files
INDEX	for generating sorted index files

REPORT to produce a printed report or text file

RMSCOPY to transfer records from one file to another

RMS.TRM descriptor file used by RMS

plus several commonly used OS-9 utilities.

The SYS directory contains the OS-9 error messages file `errmsg`, used by OS-9 when reporting errors. (for example "Error #216 - Path name not found").

You are now ready for a trial run with RMS. Try getting into RMS with:

```
OS9: rms address
```

To return to OS-9, (at any time), hit **Control**+**E**.

We suggest that you now work through the Example Application at the end of this manual, to familiarise yourself with the basic features of RMS. This will not be sufficient for you to create your own RMS applications - to do so you must read the rest of the manual carefully, section by section. RMS has many useful features which can be missed on a casual reading. Don't be afraid to try things out - it's the easiest way to learn.

Using RMS on two drives

We have described the use of RMS on a single drive, drive 0. If you have two drives, and there is room on your system disk, you will find it more convenient to move the RMS programs onto your system disk. You should copy RMS, RMSNEW, INDEX, REPORT, RMSCOPY, and RMS.TRM from the CMDS directory of the RMS disk to the CMDS directory of your system disk.

Refer to the *OS-9 User's manual* for instructions on the **copy** command.

You can now use a separate disk as you data disk, in drive 1. You could try this out by placing your RMS disk, (not the original!) in drive 1, and your system disk in drive 0. Set your execution and data directories with:

```
OS9: chx /d0/cmds chd /d1
```

Chapter 2. Data Storage under RMS

All data is stored in files on disk. Each file consists of a set of RECORDS. All records in a particular file are of the same length and have certain characteristics in common:

- all records consist of printable ASCII data,
- each record ends with a carriage return character,
- the first character of each record is used by RMS for internal categorization of the record,
- the rest of the record is divided among the various data fields as determined by the dictionary.

The user can define as many RMS data files as needed. Associated with each data file is a dictionary file, and possibly one or more index and report specification files.

2.1. The concept of a record

A record is the logical grouping of several data items into one physical entity. Data is stored and retrieved by RMS a record at a time. Usually the data items within a record will describe particular details of one useful “real-world” entity. For example, a record might contain a person's name, address, phone number, social security number and age.

2.1.1. Fields and record Keys

In the above example the particular data items (name, address, etc.) are called FIELDS. The grouping of all the fields together is called a RECORD. Each of the fields has its own particular characteristics; the social security number is always 9 characters long and must always consist of numeric digits only. The name field could consist of alphabetic characters, and so on. In RMS, it is up to the user to decide what data fields he needs, and what the characteristics of each will be. In doing so, he determines that data a record is to contain.

Normall the is on field in the record which is designated as the KEY FIELD. The key field will be the one by which immediate access to the record may be made. In the above, if the name field is designated as the key then the user can lookup and display a record by entering only the name.

2.1.2. Primary record

Within on RMS data file the user may choose to use one of two possible situations. The first situation is one in which all records in the file have the same format (ie, same set of fields). This will fit applications where the data base can be well represented by a set of individual records; one record per one real world entity. Each record in the file has a unique key field value. An example of this organization is a mailing list; another might be a stock inventory system. In this case all the records in the file are referred to as PRIMARY records.

2.1.3. Secondary record

The second possible situation is one which allows two types of records within one file. The first type of record is called the PRIMARY record; the second type is called the SECONDARY record. In this situation there may be but one primary record with any given key field value, but under each such primary record there may be any number of associated secondary records. Within the file, all primary records have the same format, and all secondary records have the same format. However, the format of the primary records can differ from that of the secondary records.

2.1.4. Record groups

When a file is of the second type (both primary and secondary records) then each primary along with its associated secondary records is called a record GROUP. A primary may have no secondary records

or may have many. A secondary record must always belong to some primary record. Some examples which help demonstrate the parent/child relationship between primary and secondary records within a group are:

1. a primary record for each customer with a secondary record for each invoice for that customer;

2. a primary record for each student with a secondary record for each class he takes;

3. patients/visits;

4. employees/pay periods;

5. ledger entry/detail item.

There are undoubtedly many other possibilities which can be considered.

2.2. Files used by RMS

The files used by RMS fill several functions. The particular function served by the file is denoted by the file suffix appended to the file name. (the file suffix is a three character code, preceded by an ".", and place after the name of a file as it is stored in the disk directory by OS-9.) The following descriptions tell what purposes each of the file types serve.

.RMS An RMS main data file is followed by the suffix ".RMS". Its purpose is to store the data records of the user's data base. It is created initially by RMSNEW. It can be updated by the RMS editor. The REPORT program reads data from it to create printed reports. It may be read also by BASIC09 programs.

.DIC Every RMS main file must have a dictionary file associated with it. The dictionary file must have the same basic file name but must have a suffix of ".DIC" when created by the user. The dictionary file is a text file which contains the information about the fields in the primary and secondary records of a file. It can be created and edited by use of the OS-9 **edit** program or the stylograph word processor. Details about the contents of the dictionary file are found in section 4.

.REP A file with a suffix of ".REP" is a text file which contains a set of REPORT SPECS. Report specs are the specific instructions about the actual format that a printed report is to have. Any number of the files may be associated with any RMS main data file. The `.REP` may be created and edited with the OS-9 **edit** program or the stylograph word processor. Details about the contents of the `.REP` file may be found section 6.

.NDX A file with the suffix ".NDX" is called an INDEX FILE. An index file is simply a text file containing a list of KEY FIELD values. It may be used to drive the editor or report program to process the RMS main data file in any desired order. There may be any number of index files for any RMS main data file. The `.NDX` file may be created by the use of the OS-9 text **edit** program or the stylograph word processor, a BASIC09 program or by the INDEX program which is part of RMS.

Chapter 3. Getting started with RMS

This section describes the general step in getting an application started using RMS. The specific details on each of the steps in the process are covered completely in other parts of this manual. It is siggested that the reader be familiar with the contents of the previous sections of this manual before proceeding, since certain terms that are used here have been introduced in those sections.

3.1. Record layout and creating the record dictionary

The first task is to examine your problem and determine what data items must be stored. Organize the data values into related sets of items. Determine whether there is a primary-only or primary-secondary group structure to your data. If there is a primary-only structure then all data fields will go in the primary record. If there is a group type structure the you must determine what items can be placed in the primary record which heads up the group, and what items must be placed in the secondary records. In general, put all common data in the primary record.

Give each data item a FIELD NAME. This name can be any word of 8 or fewer characters in length. The field names for a record might be: NAME, ADDRESS, PHONE, SOCSEC and AGE, for example. You must also determine which field is to be the key field within the primary record. The secondary records, if they are to be used, will have the same key field as the primary records.

Next determine the characteristics of each field. The first consideration is what the FIELD TYPE is to be. The choices are: Alphanumeric (any data at all), Numeric (only digits 0 to 9 allowed), Money (always stored as pounds and pence), or Data (day, month and year). In all but the case of the date, which is always 8 characters, you must determine for each field what the maximum number of characters can be. The sum of the size of the fields in the record gives you the size of the record. Lastly, you must now consider for moment how the RMS editor works.

The RMS editor builds a fill-out form on the screen for each record. To do so it places blanks on the screen which correspond to the various fields that can be entered by the operator. But for the form to make sense there must be an explanatory phrase or message next to each blank so that the operator knows what data he is to enter. The message or phrase is called a PROMPT. It is up to the user to determine what prompt he wishes to appear in front of each field on the screen. To illustrate this principle look at the following diagram. It is a representation of what a screen form might look like in the RMS editor. All the text on the form is that specified by the prompts. The dashed lines represent blanks to be filled in by the operator.

```
CUSTOMER NAME:_____ PHONE:_____
HOME ADDRESS:_____ TYPE OF PLAN:_____
TOWN:_____COUNTY:_____ POSTCODE:_____ CONTACT DATE:_____
CURRENT BALANCE:_____ CREDIT LIMIT:_____ STATUS:_____
```

Once the field characteristics have been determined, a dictionary file must be created. This is accomplished by coding the field names, types, lengths and prompts into a text file. The name of the text file must correspond to the name to be given the RMS data file and must have a suffix of ".DIC". See section 4 for more details. The dictionary file is created with the OS-9 text **edit** program or the stylograph word processor.

3.2. Formatting the RMS main data file

Once the record dictionary has been created, the **.RMS** file can be created. The **.RMS** file has the same root name as the **.DIC** file but has the suffix ".RMS". These files should be stored in the same directory on disk. The **.RMS** file contains the actual data records. The create the **.RMS** file you use the RMSNEW utility program. It is stated by entering:

```
OS9: rmsnew name
```

The name may be any properly qualified OS-9 file specification. The **.RMS** suffix will be added on automatically and should not be entered. Before proceeding, the utility will request two values. The first is the record length. This must be a number which is large enough to accomodate the longer of the primary and the secondary records to be stored in the file. The possible record length may be any number from 3 to 1022 characters. It is permissible for a record length to be too long to accomodate the desired records but not too short. RMSNEW will add two to the entered length: one for a prefix identifier byte on each record, and one for a return character at the end of each record.

The second number which must be entered is the number of records to be stored in the file. Because of the hash coding technique used by RMS to store and retrieve records, this number should be at least 25% more than the actual maximum number of records that will ever be stored in the file. The reason is that hash coding only works rapidly if the target file is less than about 80% full. Also advantageous but not necessary is to make the size of the file a prime number. This is not essential but can improve the access time average for the data records in the file. The maximum number of records allowed is 65535.

3.3. Putting Data Into the File

Once the **.DIC** and **.RMS** files have been created, data can be entered into the file using the RME editor program. The complete details of its use are found in section 5, but the general steps to add one or more records are:

```
enter the command...
OS9: rms name
```

Where “name” is the same name entered when creating the file with RMSNEW. RMS will add the suffix “.RMS” to locate the main data file and “.DIC” to locate the dictionary.

Now fill out the displayed form with the desired data, use the INSERT function key to put a new record into the file.

3.4. Modifying Data in the File

To change the data in one or more records of the file:

- Enter the RMS command shown above,
- fill in the key field of the desired record and push the FIND function key to display the data for that record,
- change the desired fields in the record.
- Use the UPDATE function key to put the modified record back in the file.

3.5. Getting Data Out of the File

There are two ways to get data out. The first is to use the RMS editor to find and display the desired record as it is needed. The second is to use the REPORT program to produce a printed report or text file with all or the desired portion of the data from the file. The steps to create a printed report or text file are covered in detail in section 6. However, in general the process is:

- create a report spec file indicating what the report is to look like,
- then run the REPORT utility program by entering:

```
OS9: report name sfile
```

where “name” is the name of the data file (without the “.RMS” suffix) and “sfile” is the root name of the report spec file.

3.6. Use of an Index File

In the simple case an index file is used to create a report in a sorted order by one of the fields. This is accomplished by running the INDEX program to create a **.NDX** file; then the report spec can designate that the **.NDX** file is to be used to force the desired order on the output of data records. More on the use of index files may be found in section 7.

Chapter 4. The RMS dictionary

The dictionary is stored in a file with the same root name as the RMS main data file but with a suffix of ".DIC" added to the name. The purpose of the dictionary is to describe the layout of the fields within the primary record and, if one is to be used, the secondary record. The **.DIC** file is formatted as a text file, and may be created with the **edit** program or the stylograph word processor. It should be stored in the same disk directory as the **.RMS** file. As an example, a dictionary might look like the following:

```
    "EMPLOYEE BENEFITS "
NAME    20    A    "EMPLOYEE NAME:" ;
SOCSEC   9    N    "SOCIAL SECURITY NUMBER:";
DEPT     3    A    "DEPARTMENT";
SAL      7    M    "SALARY:" ;
PLAN    15    A    "BENEFIT PLAN: " ;
HDATE    8    D    "DATE HIRED:" ;
$
```

The first seven lines above are called a record specification. The top line is called the TITLE LINE. It is used by the RMS editor. The text within the quotes will be printed on the second line of the screen to indicate to the operator what form is being displayed. The next 6 lines represent 5 field specs with the record. Each of the fields is given a FIELD NAME. These names are arbitrary but are usually chosen as meaningful words. After the field name is a number which determines the maximum number of characters that can be entered into the field. Next comes a one letter FIELD TYPE CODE. This code determines the type of data which is to be stored in the field. The type codes shown are A(lphanumeric), N(umeric), M(oney) and D(ate). Next, each line has a PROMPT, which is a text string in quotes. The RMS editor uses these strings to display a meaningful form to be filled out by the operator. Each prompt appears on the screen preceding a blank space long enough to hold the corresponding field. Finally, each field line in the dictionary **must** be terminated by a semi-colon.

The details on building a dictionary are presented below, but in general:

- Each item on a line is separated from the next by a comma, or by spaces, or both.
- The first field name given is assumed to be the KEY FIELD of the record.
- Any field spec may be continued on another line; to do so, just break it a comma or a space, and continue on the next line.
- Each field spec **must** be terminated by a semi-colon.

4.1. Title line

The first line of a record specification must be a title line. It consists of two quote marks (") with from 0 to 80 characters between. The RMS editor displays the text on the second line of the screen whenever the form for the the record is displayed.

4.2. Field specs

After the title line, there may be one or more field specs. Each one represents one data field in the record. The first field spec represents the KEY FIELD of the record. There may be up to 50 field specs within one record specification.

4.2.1. Field name

The field name is the first item in each field spec. It may be any word of the user's choosing which is from 1 to 8 characters in length. The field names used within on file should all be unique. Letter case in field names is ignored, so that **PHONE**, **Phone**, and **phone** are all the same name.

4.2.2. Field length

The field length is the number of characters reserved within each data record for this field. The minimum length is 1. The maximum field length is 255 characters. A date field must always be length 8. A money field must be at least length 4. Other than these restrictions, the user may chose whatever lengths he sees fit.

4.2.3. Field type

The field type code is one letter, in upper or lower case. The RMS editor will insure that only valid data is keyed into any field according to the type code given. The possible types of fields are:

A	Alphanumeric	Any data may be stored in this field
N	Numeric	Only digits from 0 to 9 may be stored in this field. Also, before storing any record, all data in N type fields will be adjusted with the field to insure right-justification. If necessary, the field will be padded on the left with spaces to attain right justification.
M	Money	Only digits 0 to 9 and a decimal point may be stored in this type of field. Also, before storing any record, all data in M type fields will be automatically adjusted to proper pounds and pence form. Specifically, there will always be exactly 2 decimal places and the number will be right justified within the field.
D	Date	A date field may only contain a value in the form of DD/MM/YY. DD is the day from 1 to 31, MM is the month from 1 to 12. And YY is the year from 00 to 99. Before storing any record, all D type fields will be adjusted to insure the standard data format. The editor also provides some time-saving short cuts when entering data into a data field, by automatically inserting the "/" characters as you enter the data.

4.2.4. Optional fill-in flag

Unless otherwise designated, the RMS editor will not store a data record into the main file unless all fields within the form are filled in and validated as to type and validators. However, if the user wishes to designate that the operator may leave a field blank within a record this is done by entering an "optional fill-in flag". This consists of an asterisk "*" immediately following the type code. For example:

```
SALARY    7    M*     "MONTHLY SALARY" ;
```

If the * is used then the operator may leave a field completely blank when inputting a data record with the RMS editor. However, if he keys any data other than blanks into the field, then the field must conform to all the type and validator restrictions.

4.2.5. Prompt

After the field type code comes the prompt value. The prompt consists of a pair of quote marks (") with any number of characters in between. The prompt is displayed on the screen by the RMS editor when the fill-out form for this record is displayed. An empty prompt, signified by two adjacent quote marks, indicates that no text is to be displayed preceding the blank for this field. The prompt may carry over for as many lines as necessary as long as it is ended with a quote mark. Skillful use of placing blanks withing prompts will allow the user to force the screen to look as desired. In general, it will be easiest to do this by trial and error. A few attempts at creating a form then running the RMS editor to see what it looks like will quickly show how the editor uses prompts to build the form. Essentially, the editor always ensures that the prompt and the associated data space are on the same line, provided that their total length is less than the line length.

4.2.6. Validators

In addition to the above items for each field spec, it is possible to further restrict the allowable values for any particular field. This is done by the use of one of three types of VALIDATORS. Validators are very useful in assuring that only “correct” data is entered into the file using the RMS editor. The editor will not store a record into the file if the data in any field violates the restrictions placed on the field by the validators. The optional-fill-in flag be used to designate that a field may be left blank and thus not conform to the validator.

A validator, if used, is placed after the prompt item, and before the semi-colon. There may be a most one validator per field spec but the may be any number within the record specification.

4.2.6.1. Minimum length validator

The first type of validator is the minimum length validator. It is coded as follows:

```
NAME   20   A   "STUDENT NAME"  <10> ;
```

The **<10>** denotes that there must be at least 10 non-blank characters in the NAME field for it to be considered valid.

4.2.6.2. Range validator

The range validator is coded as follows:

```
PRICE   6   M   "RETAIL PRICE "   (  2.00,599.99) ;
```

The range validator denotes that, to be considered valid, the data value must be greater then or equal to the first item in parentheses and less than or equal to the second item in the parentheses. In this case the value would have to be at least 2.00, and no more than 599.99. **IT IS VERY IMPORTANT TO NOTE THAT** the items in the parentheses must each be exactly the same length as the length of the data field, and must be in the form to which the editor would automatically convert for date, numeric or money fields. Failure to observe this rule will result in incorrect data validation. The minimum and maximum values must be separated by a comma.

The comparison is arithmetic for money or numeric type fields. For alphanumeric fields the comparison is a left to right character comparison determined by the ASCII code. In general this produces alphabetic order - “A” less than “B”, etc.

For a date field the comparison is chronological. That is, a date that is earlier is considered to be less. For example 25/08/85 is less (sooner) than 30/01/86.

4.2.6.3. List validator

A this type of validator is the LIST validator. This provides a means to explicitly list all the possible values that are to be considered valid. The format is shown by this example:

```
GRADE   1   A   "FINAL GRADE FOR THE CLASS:"  [A,B,C,D,F]  ;
```

The items within the brackets are considered to be a complete list of the possible valid entries for the field called GRADE. The list may have any number of entries and may carry on for as many lines as necessary as long as it ends with a “]”. The items must be separated by commas as shown. Each item should be exactly the same length as the data field, and should be in the format to which the RMS editor will automatically convert for money, numeric or date fields. If a list item is longer than the data field it will never match. If is is shorter than the data fields then the data field is only checked for a match up to the length of the list item. If is is equal up to that point, then it is valid regardless

of the value of the rest of the field. LETTER CASE IS NOT IGNORED IN THE MATCHING - thus “a” does not match with “A”.

4.3. Primary and secondary record specifications

If only primary records are to be stored in the file then only one record specification is entered in the dictionary file. It may be ended by a $ as shown in the example in section 4 or the $ may be omitted. If there is to be a secondary record specification for the file then it is separated from the primary record specification by a $. After that is is formed in exactly the same manner as the primary record specification. For example:

```
"CUSTOMER MASTER RECORD"
NAME  20   A   "CUSTOMER NAME" ;
PHONE  7   N   "PHONE NUMBER"  ;
CREDIT 6   M   "CREDIT LIMIT"  ;
OUTSC  6   M   "OUTSTANDING CREDIT BALANCE";
PDATE  8   D   "LAST PAYMENT DATE" ;
$
"CUSTOMER INVOICE RECORD"
NAME  20   A   "NAME" ;
IDATE  8   D   "INVOICE DATE" ;
AMT    6   M   "INVOICE AMOUNT" ;
ITEM  15   A   "ITEM DESCRIPTION" ;
$
```

The primary record is entitled "CUSTOMER MASTER RECORD", and the secondary record is entitled "CUSTOMER INVOICE RECORD". This dictionary would allow a file with one primary record for each customer name, optionally followed by any number of secondary records, one for each invoice. All the rules for building the primary record specification apply to the secondary record specification as well. In addition there are a few more rules:

1. The key field (first field) in the secondary record must have the same field name, length and type code as the primary record. The prompt may differ.

2. The primary record spec could have but one field if desired. The secondary record spec must have at least two fields, (including the key field).

Note that all the user need to do to use secondary records in a file is to include the secondary record specification in the dictionary.

Chapter 5. The RMS editor

The RMS editor is used to add records to the RMS main data file; display the contents of records in the file; change the contents of records or remove records. To run the RMS editor type:

```
OS9: rms name index
```

"name" is the root name of the **.RMS** and **.DIC** files. If the "index" file is to be used then it must have the suffix **.NDX**. The index file is optional; its use is discussed later. A sample command to start the editor is:

```
OS9: rms patient names
```

This command would assume that PATIENT.RMS and PATIENT.DIC reside in the current directory and that NAMES.NDX does as well.

In general RMS displays a form on the screen. It is made up of the title line and the various prompt strings and blank areas for the data fields. The top line of the screen is reserved for any necessary messages from RMS to the operator. The second line will contain a title. The rest of the screen has a prompt and a blank area for each field of the record. Depending on what is going on, the screen may be displaying the primary form or the secondary form.

Once the form is displayed the user can fill out the form and fetch or store records into the data file.

5.1. A screen fill out form

The form on the screen represents a picture of what a paper form for the same data might look like. The cursor will only come to rest within the blanks on the form where the operator should enter data. AS one field is filled the cursor will skip to the start of the next field. The prompt strings that were define by the user and placed in the dictionary will, if designed carefully, always inform the operator as to what data is to be entered into each field. The type of each field, also designated in the dictionary, will determine exactly what data may be entered into a field. For example, if the cursor is in a numeric field then the only printable characters that can be entered are space and the digits 0 to 9. If any other key, such as "A"; is pressed then the audible alarm on the terminal will beep, indicating an invalid key.

5.2. Putting data into a form

To enter data into a field within the form it is necessary to first move the cursor to the desired field and position with the field. There are several comman keys that move the cursor. Refer to the Key Assigments section at the end of this manual for the keys used on your keyboard.

RETURN	moves the cursor to the start of the next field, if in the last field the cursor is moved to the first field.
RIGHT-ARROW	acts in the same manner as RETURN; it can be thought of as a TAB key.
LEFT-ARROW	is a BACK TAB; the cursor is moved to the start of the previous field on the screen. If the cursor is in the first field on the screen then no action is taken.
HOME	moves the cursor to the first field on the screen.
BACKSPACE	moves the cursor one position to the left and erases the character under the cursor. It may be used to correct typing errors while entering data.

CLEAR clears the screen and displays a blank primary form.

If a secondary form is displayed the editor will automatically fill in the key field (first field) with the key value of the associated primary record. Furthermore, the cursor will never come to rest in the key field of a secondary record since it may not be changed. Whenever the cursor would otherwise go to the key field (such as in response to the HOME key) it will go to the second field instead.

Once the cursor is in the desired position data may be entered into a blank field, or an existing field may be modified by simply typing on the keyboard, overwriting the existing characters. If a key is typed which is not valid for the context then the terminal will beep and no action is taken. **REMEMBER** also that any new data entered on the screen is not saved in the disk file until either the INSERT or UPDATE keys (described below) are pushed.

When the cursor is in an alphanumeric field any printable characters may be entered. When in a numeric field only space and the digits 0 to 9 may be entered. When in a money field space, digits 0 to 9 and a decimal point may be entered. It is not necessary to put in a decimal point if the money value is in whole pounds but the must be enough room for the decimal point and two zeros to be added (see RECORD VALIDATION below). When in a data field the date must be finally stored in the form DD/MM/YY, (day, month and year). However when entering a date the operator can save keystrokes on a couple of ways. If the month or day is one digit then it is not necessary to put in a leading zero. Just put in the one digit then push "/" or the space bar and RMS will fill in the leading zero and the "/". Or, if the month or day is two digits then it is not necessary to put in the "/" or push space. When the next digit is entered RMS will fill in the "/" automatically. This all seems confusing but if you play with it a bit you will see that it is really quite convenient.

5.3. Command keys in the editor

All actions concerning the fetching and storing of records in the data file are invoked by sngle key commands. If for any reason a command function cannot be executed or a message to the operator is necessary then the message is displayed on the top line of the screen. Once a message is displayed the user is asked to push the RETURN key to go on. Once the key is pressed, the message goes away and the user may then proceed as desired. The command keys and their actions are described here. Refer to the Key Assigments section at the end of this manual for the keys used on your keyboard.

QUIT causes the termination of the RMS editor and immediate return to OS-9.

FIND finds the primary record in the file that has the key value which is now displayed in the key field (first field) on the screen. Then the record is displayed in the rest of the form. The basic use of FIND is to enter the key value of a record, then push FIND to display the rest of the record in the form. Also, whenever a secondary form is displayed on the screen, pushing the FIND key will re-display the associated primary record.

CLEAR clears the screen and displays a blank primary form. If a secondary for is displayed on the screen and you wish to use the SCAN key, you must first push CLEAR to get a primary form displayed.

INSERT causes a new record to be placed in the data file. All fields will first be checked for valid data as described under RECORD VALIDATION below. If any field is found to be invalid the record is not written to the file. The operator may correct the problem then try INSERT again. If a primary form is displayed then a new primary record is created. The key field as displayed in the form must not already be in the data file. If it is, then the new record is not written to the file. If a secondary form is displayed then a new secondary record is added to the end of the current record group.

UPDATE is used to change the contents of one or more of the fields of a record. To use UPDATE the record must first have been displayed by the FIND or SCAN keys for a primary record or by the FEED key for a secondary record. Once displayed, any fields, except the key field may be modified. Then UPDATE will cause the updated record to replace

the previous record in the file. Before the record is written it is checked for valid data as described below. If any field is found to contain invalid data then the record is not rewritten to the file.

DELETE is used to remove a record or group of records from the file. Once a record is removed is it not retrievable. For this reason whenever DELETE is pressed a message is displayed on the top of the screen asking if it is OK to delete the record or record group. The operator must respond with a “D” to continue with the delete. Any other key causes the command to be ignored. Before deleting a record it must first be displayed by the FIND or SCAN key for a primary record, or the FEED key for a secondary record. An attempt to delete a primary record will also delete all secondary records in the group (if there are any). If a secondary record is displayed when DELETE is pressed then only that particular record is deleted.

FEED is used to display the next secondary record in the current group. It is necessary to first have displayed the primary record with the FIND or SCAN key before any secondary records may be displayed. Each time FEED is pressed the next secondary is displayed. It may then be updated with UPDATE or removed from the file with DELETE. If it is deleted then the next secondary will be displayed when the delete function is complete. At any time when a secondary record is displayed the FIND key may be used to re-display the associated primary record at the beginning of the group. When there are no more secondary records in the group the FEED key will cause a blank secondary form to be displayed with the key field filled in to reflect the key of the group. To get a blank primary form push CLEAR. If there is no secondary record specification in the dictionary then the FEED key may not be used.

SCAN is used in one of two ways, depending on the presence or absence of an index file when the RMS editor was invoked. In either case it may only be used when a primary form is displayed.

If no index file was specified then SCAN is used to browse through the file. Each time SCAN is pressed a sequential search is made in the file for the next primary record to be found. Then that record is displayed on the screen. Due to the hash coding technique the records will be located in a random order. It may also be possible that no record is found quickly since records can be spread out within the file. For this reason, if no record is found within a couple of hundred slots or the previous one then the search stops and a blank primary form is displayed. If you wish to continue just hit **SCAN** again. When all records of the file have been displayed a message is displayed on the top line telling the operator that further SCAN commands will start over from the beginning of the file.

If an index file was specified on the command line that started RMS, then SCAN is used in an alternate manner. The index file is assumed to contain a list of key field values, one per line. The specific details on how to create an index may be found in section 7. Each time SCAN is pressed the next key value is read from the index file and displayed in the key field of the screen. Then the equivalent of a FIND key command is executed. That is, the record is found in the file and displayed in the screen form. If no record exists with that key then a message is displayed on the top line of the screen. Note that is a record exists in the data file but its key field is not in the index field, it will not be found by SCAN. It can only be found by FIND.

Once SCAN is used to display a primary record it may be updated by UPDATE, or removed by DELETE, or the associated secondary records may be displayed by FEED.

DUPLICATE the contents of the field in which the cursor is resting are duplicated from the last record of the same type (primary or secondary) that was written to the file by an INSERT or UPDATE. DUPLICATE can be useful if several records are to be entered and one or more of the fields need to be the same in each record.

5.4. Record validation

Each time the operator tries to write a new record into the data file by pressing INSERT or UPDATE, the contents of the screen form are automatically checked for validity. First of all, the data must conform to the field type. A data field must have a reasonable data; ie, the month must be from 1 to 12, etc. If necessary minor adjustments are made automatically. These might include right justifying a numeric or money field, supplying any missing decimal places in a money field, and so on. All such changes are shown on the screen as they are made. The second type of checking is that called for by any of the four types of validators that the user may have entered in the dictionary. See section 4 for more details on validators. If for any reason a field is found to be invalid, the appropriate message is displayed on the top line. The operator must press space to go on. Once he does so the message is erased and the INSERT or UPDATE is discontinued. The error may be fixed and the INSERT or UPDATE retried.

The optional-fill-in flag described in section 4 may be used to specify that a field may be left blank. In this case, a blank field is considered valid even if it would not otherwise be valid. If the optional-fill-in flag is not set in the field spec then the field must conform to all field validations. If the record is valid then it is written to file as requested.

Chapter 6. The RMS report writer

The REPORT program is used to create a printed listing of all or part of the data in the RMS data file. Through the use of the various report commands a wide variety of options may be selected to build customized reports. A report might consist of a simple listing of the records, a set of mailing labels, a ledger, trial balance, inventory or stock report, reorder lists, checks, receipts, or any number of different management information reports. REPORT is also used to create disk text files for use by other programs, for instance a list of names and addressess to be used by the MAIL MERGE program.

6.1. How to use REPORT

To use the REPORT program it is necessary to first create a REPORT SPEC file. A report spec file is simply a text file with the necessary report commands to specify what the report is to contain and how it is to look. A report spec file may have any root name but must always have a suffix of ".REP". Any number of **.REP** file may be created and re-used as many times as necessary. The following sample command illustrates the use of REPORT.

```
OS9: report client weekly
```

In this example, the current directory contains the main data file, CLIENT.RMS, and the associated dictionary CLIENT.DIC. The file WEEKLY.REP is the report spec file. The report printout will be written to the standard output, normally the screen.

6.2. What REPORT can do

Using the report commands described below, the user may choose any combination of the following features.

a. report title page before the body of the report.

b. report wrap up page after the body of the report.

c. process the records in physical order or in any chosen order as driven by an index file.

d. adjust to any page size or special form layout.

e. automatic skipping over page perforations.

f. selection of specific records to be part of the report, by low/high bounds or by set selection.

g. selection of specific records to be excluded from the report by low/high bounds or by set exclusion.

h. automatic new page for each primary record or each secondary record (or both, or neither).

i. print any number of lines for each selected primary record.

j. print any number of lines for each selected secondary record.

k. automatic page overflow headers, any number of headers allowed.

l. end of record group print lines to summarize information from each record group.

m. automatic record counting by group or by file.

n. automatic totalling of numeric and money fields available as file totals, total so far, or subtotal by record group.

o. current date and page number available.

p. user specification of exact format of each type of print line.

6.3. Routing output from the report writer

The REPORT program writes all output ti the standard output file. This means that the output will normally come to the screen but may be redirected to a disk file or printer. For example, to send the output to the parallel printer port “/p”:

```
OS9: report client weekly >/p
```

or to send to the serial printer port “/p1”:

```
OS9: report client weekly >/p1
```

or to send the output to a text file on disk called “OUTFILE”.

```
OS9: report client weekly >outfile
```

When the REPORT command is given it is assumed that the print mechanism is positioned at the first print line that is to be actually printed upon a new page.

6.4. The report spec file

The commands described below are entered into a file with a suffix of “.REP”. This may be done with the OS-9 **edit** program or the stylograph word processor. Each command *must* be terminated with a semi-colon. Any particular command may carry over as many lines as necessary as long as it is broken between print line items. In general, the report command may be entered into the report spec file in any order. The only case in which the order matters is the case of multiple print lines of the same type. The types of print line commands are T, W, P, S, H and G. For example, if there are three T commands then they are processed in the order that they are found in the report spec file.

6.4.1. L command - lines per page

format: `L x,y ;`

x and y represent numbers. The L command tells how many print lines there are on a page, “x”; and how many lines of those are to be actually printed upon, “y”. “y” should be less than or equal to “x”. If no L command is found then the page size is assumed to be 66, and the printed lines per page is 60. REPORT will automatically print blank lines to cross page boundaries. A command of “L 1,1 ;” will have the effect of nullifying all automatic skipping across page boundaries. The is the form that must be used to create unpaginated text files, for example for use by MAIL MERGE.

6.4.2. I command - include

format: `I fieldname (low,high) ;`
or `I fieldname [value,value,value,.....] ;`

The I command allows for selection of the desired records for inclusion within the report output. In both forms the field name may be that of a field in the primary record or the secondary record. The first form allows for accepting records only if the named field has a value greater than or equal to the low value, and less than or equal to the high value. The values in the parentheses MUST be exactly the same length as the fields and must have the data coded in exactly the same manner as it will be found in the field. For example, a date field should be in DD/MM/YY form, a money field must be right justifiedm oadded with spaces on the left if necessary, and must have exactly two decimal places. The second form allows for inclusion of records only if the named field has one of the values in the

list. The values in the list should be exactly the same length as the field and should follow the same rules just stated. If a list value is shorter than the designated field then the field need only match for the given item's length. In both cases the use of the “include” is analogous to the use of a validator in the RMS editor (see section 5.4). Any number of “include” or “exclude” commands may be given. For a record to be selected as part of the report it must conform to all selections specified. If a primary record is excluded due to an I or an E selection then the entire record group is skipped. If a secondary record is excluded then there is not any automatic effect on any other records in its group.

6.4.3. E command - exclude

format: `E fieldname (low,high) ;`
or `E fieldname [value,value,value,.....] ;`

The E command allows for selective exclusion of specific records from the report. It works just like the I command except that to be accepted for the report a record must NOT be within the specified bounds (first form), or must NOT be in the specified list (second form). As stated above, to be accepted as part of the report a record must conform to all I or E commands.

6.4.4. T command - title

format: `T print-line ;`

The T command specifies a line to be printed on the report title page. There may be any number of T commands. If there are none then there will be no title page. The title lines are printed about one quarter of the way down the title page. After the title lines are printed the form is moved up to the first print line of the next page.

6.4.5. W command - wrap-up

format: `W print-line ;`

The W command specifies a line to be printed on the report wrap-up page. There may be any number of W commands. If there are none then there will be no wrap-up page. The wrap-up page is printer on a new page at the end of the body of the report.

6.4.6. P command - primary records

format: `P print-line ;`

The P command specifies a line to be printed for each primary record that is included in the report. There may be any number of P commands. If there are more than one then all P commands are processed, in the order they are found in the report spec file, for each primary record. If no P commands are corded then no lines are printed for each primary record.

6.4.7. S command - secondary records

format: `S print-line ;`
The S command specifies a line to be printed for each secondary record that is included in the report. There may be any number of P commands. The may be any number of S commands. If there are more than one then all S commands are processed in the order they are found in the report spec file, for each secondary record. If no S commands are corded then no lines are printed for each secondary record.

6.4.8. H command - page header

format: `H print-line ;`

The H command specifies a page header line to be printed on the top of the page in the case that the page becomes full and a new page skip is automatically done. There may be any number of H

commands. They are processed in the order they are found in the report spec file. Note that the H command specifies a page header line only for the case of page overflow. If a new page is reached as a result of the page break (B command) then the H commands are not processed. In the case of a new page due to a B command, the P or S commands can be used to print any page headers that are necessary.

6.4.9. G command - record group

format: `G print-line ;`

The G command specifies a line to be printed at the end of a group of records. If there are primary and secondary records in the group then the G commands are processed after the last secondary record of the group. If there is only a primary record in the group the G commands are processed after that record. The G command is used to summarize information that pertains to a group of records. Any number of G commands may be specified.

6.4.10. B command - page break

format: `BP ;`
or `BS ;`

The B command specifies a page break for the primary or the secondary record type. If a BP is coded then a new page is started for each primary record processed. If a BS is coded then a new page is started for each secondary record processed. One or both of these commands (BP and BS) may be coded in one report spec.

6.4.11. X command - index file

format: `X filename ;`

The X command us used to specify the name of the INDEX file to be used by the REPORT program. The index file is a text file which contains a list of record keys. The name of the file is assumed to be “filename.NDX”, the “.NDX” is not actually entered in the X command.

Each line of the text file must contain a record key field value. If the is other data on the line after the key field value it is ignored. The record groups in the data file are selected and processed in the order of the keys in the index file. The index file can be a list of all or some of the keys and may be in any desired order. Records whose key fields are not found in the index file will not appear in the report. The index file may be created by the INDEX program described in section 7 or by a BASIC09 program or text editor.

A special use of the X command is to specify the index file to come from the keyboard (standard input):

```
X <prompt ;
```

The “<” symbol indicates to REPORT that the string is a prompt, and not a filename. Now each time REPORT is read to process a new record group it will display the prompt string on the screen and wait for the operator to enter a key field value, followed by RETURN. If RETURN or ESCAP is the first key hit, REPORT takes this to mean that no more records are to be processed.

This mode of operation is very useful for interactively displaying or printing record information, for example printing address labels - the operator can ask for none, one, or any number of records to be printed.

If no X command is coded in the report spec file then the record groups within the file are processed in the physical order they are found in the file. This will be a random order due to the hash coding technique used to locate the records.

6.5. How to make up a print line with the REPORT writer

Several of the REPORT commands call for a “print line”. A print line is a list of values to be printed on one line of a report printout. The general form of a print line is:

```
item item item     ;
```

That is, there may be any number of items. Each one is separated from the previous one by one or more spaces. A print line which contains no items designates a blank print line. A semi-colon must appear after the last item.

The items that can occur in a print line are:

field-name	the contents of that field are printed
"any text"	the text between the quotes is printed
$D	today's date as DD-MM-YY is printed
$P	the current report page number is printed as a 4 digit number
$T	the total number of selected records in the report is printed as a 5 digit number
$G	the total number of selected groups in the report is printed as a 5 digit number. This can also be thought of as the number of primary records printed since each group is associated with one primary record.
$S	the total number of selected secondary records in the current record group is printed as a 5 digit number.
field-name#n	the arithmetic sum of the contents of the named field in all records is printed as an n digit number. “field-name” must identify a field of type N or M. This total field is printed with its value so far, and it accumulates all the way through the report. Normally, this is used to total up a field for the entire report and print the value on the wrap-up page, but it can be used to print a running total with each record.
field-name%n	the arithmetic sum of the contents of the named field in all records is printed as an n digit number. “field-name” must identify a field of type N or M. This subtotal field is cleared to zero after each time it is printed. Normally, this is used to total up a field within a record group and print the value in a group-end line.

For the counter ($T, $G, $S) values, and the total (...#.) and subtotal (...%.) values, only the selected records are counted or totalled. That is, any record which is omitted due to an I or E command, or because its key field is not in the index file (if one is used), is not included in the count or total.

The total field, which is specified by a field name, followed by a #, followed by a number, is used to add up the contents of a particular field over an entire file. It makes most sense to include such a total in a “W” print line. This would print the total on a wrap up line. But, the total could be place in any type of print line. It would then give the running total so far in the file.

MISSING

Column designator - @

Following each of the print items there MUST be a column designator. The format is “@n”, where n is the column number, from 1 to the line length. This number indicates the column in which the print item is to start.

A few sample print lines will demonstrate.

```
G  "MONTH END REPORT"@10  $D@30  "PAGE"@50 $P@55 ;
```

might produce a line from a primary record like...

```
JOHNNY Q. DOE       332456223   856.50
```

```
W $S@1 "CHECKS  WRITTEN TO"@7  $G@25  "VENDORS.  TOTAL CASH OUT IS"@31
AMOUNT#7@58 ;
```

might produce a line in the wrap up page like:

```
   34 CHECKS WRITTEN TO    15 VENDORS.  TOTAL CASH OUT IS 3245.55
```

When including a field name in a print line, the value printed is the one from the last record of the appropriate type that was passed. For example, in an “S” command, which prints a line for each secondary record, if a field name is specified for a field in the primary record, then the value printed is the value from the last primary passed. In this case it would be the primary at the head of the same record group.

A useful thing to know is the REPORT formats the line by placing the items in the print buffer in the order they are found in the command. Then the entire print buffer is printed. This means that the items in the print line command new not bi in any particular order. It also means that it is possible to overlay all or part of an item by a later item in the line. The following example shows how to use this fact. Remember that the secondary record count field “$S” is printed as a five digit number. Assume we know that the actual count can only be a three digit number, because we will never enter more than 999 secondary records in a group. The following print line includes this counter but leaves only three spaces for it to print by overlaying it with a later literal.

```
W  $S@32  "THE NUMBER OF OVERDUE NOTICES IS"@1 ;
```

would print...

```
THE NUMBER OF OVERDUE NOTICES IS 512
```

Chapter 7. INDEX files with RMS

The RMS editor and the REPORT writer can each be driven by an optional INDEX file. In the RMS editor, the INDEX file controls which primary record is to be displayed on each successive use of the SCAN function. This allows the operator to view and/or change a pre-determined set of records in a given order. In the case of the REPORT program an INDEX file can be used to specify which primary records are to be processed, and to determine the order of processing. This the report can include any set of the records in the file, and can be printed in any desired order.

Typically, an index file will have the effect of sequencing the records into ascending order by one of the fields within the records. It is possible, however, for an index file to impose any arbitrary ordering on the file. The index can include all records, or can leave out records, or can repeat any record key causing, it to be processed more than once.

Any number of index files can be created and associated with an RMS data file. An index file can have any root name but must have a file extension of ".NDX".

7.1. Format of an index file

An INDEX file is just a text file which contains one record key field value on each line. The key field value should match one that is found in a key field (first field) of a record in the data file. Letter case is ignored in the matching. (Thus SMITH and Smith will match). It does not matter what other data is after the key field in the index file, as long as the key field starts in column 1 of the line and the other data is after the full key field length, with spaces in between. It is not necessary to hace trailing blanks after a key value that is shorter than the key field as long there is no other data after it on the line.

7.2. Creating an index file

There are many ways to create an index file. The simplest and most convenient way is to use the INDEX program supplied with RMS. However, it can be done with the OS-9 **edit** program or the stylograph word processor. In the case the key values are just entered as text, one per line. Another method is to write a file of key values from a BASIC09 program. The index file is produced as a sequential file with one key value put out by each PRINT. (To do so, however, it is necessary to select only records from the RMS file that have a "1" in the first byte. This will cause only primary records to be sorted, and will exclude secondary records, deleted records and unused records. More information on the format of the RMS file may be found in section 9.

7.3. The RMS INDEX utility

Included with RMS is the INDEX utility. It can be used to create a ".NDX" file with one command. The index file can be ordered by any field in the primary record. To use INDEX enter a command like the following:

```
OS9: index mailing postcode post
```

In this example, the main file is "MAILING.RMS". The index file to be produced is to be called "POSTCODE.NDX", and the field which is to specify the order of the index file is called "POST". There are always three arguments after the INDEX command. The first is the name of the RMS data file. The associated dictionary file must also be present in the same directory as the RMS data file. The second argument is the name of the index file to create, without the ".NDX" suffix. If the ".NDX" file already exists then the old one is deleted and replaced by the new one. The third argument must be the name of a field in the primary record specification. It need not be the key field, any field may be used. The INDEX file produced will always contain a complete list of all key field values in the file. The ordering of the list will be determined by whatever field name is given in the INDEX command.

It may be necessary for the INDEX utility to create one or more disk files for scratch pas use while it is sorting the data file. If so, the are created in the current directory. The temporary files will have a root name of “TEMP”. Unless a problem forces an untimely end to the INDEX program, the scratch files will always be deleted before the program ends.

If the INDEX program reports that it cannot sort the file because it is too large, or if the individual records in the file are large, try giving INDEX more working memory:

```
OS9: index mailing postcode post #15k
```

Chapter 8. RMSCOPY utility

The normal **copy** program should be used in preference to RMSCOPY when making a simple backup copy of an RMS dara file, as it is quicker.

To use RMSCOPY enter the following command:

```
OS9: rmscopy source destination
```

The source is the root name of the ".RMS" file to be copied from. The destination is the root name of the ".RMS" file to be copied to.

The RMSCOPY utility is provided as an aid in managing RMS data files. It performs a copy, record by record, from one RMS data file to another. Each record is read from the source file and inserted into the destination file. This allows a file to be copied into an empty file or into a file that already has records in it. In the latter case, the previous file contents are preserved. Both the source and destination files must have been created with the RMSNEW program, and each must have a dictionary file in the same disk directory as its associated RMS data file. If the two dictionaries are the same then each record is copied as is. If the dictionaries are difference, then the destination records are built field-by-field according to the following rules:

- If a field name exists in both dictionaries, then the field data from the source file is copied to the corresponding field in the destination file.
- If in the above case, the field lengths differ, then the source field is truncated or padded with spaces to fit the destination field. The truncation or padding occurs on the left end for numeric or money fields, and on the right for alphanumeric fields.
- If a field name occurs in the source dictionary but not in the destination dictionary, then the contents of the field are not copied anywhere.
- If a field name occurs in the destination dictionary but not in the source dictionary, then the field is left blank in the destination file.
- The key field name of the destination dictionary MUST be present in the source dictionary.

After each record is constructed it is inserted into the destination file. If the key field value of the new record is found to be already in the destination file, then the new record is not copied into the destination file. The key field value of the record will be displayed on the screen, to show that this has happened.

8.1. To change the size of a file

The size of an RMS data file can be changed by these steps. For explanation purposes the original file name is XXX.

- Use RMSNEW to create a file of the desired length, call it YYY
- Copy the dictionary file from XXX.DIC to YYY.DIC, using the OS-9 'copy' command
- Copy the data from XXX.RMS to YYY.RMS using:

  ```
  OS9: rmscopy xxx yyy
  ```

- YYY could now be renamed to XXX if desired, after deleting XXX if it is in the same disk directory

8.2. To change the record dictionary for a file

It is possible to re-arrange the fields within the records, add new fields, delete fields or change the length or one of more fields. It is even possible to change the field that is to be the key field. For example: the old file is keyed on “name” byt contains social security number as a data field. The new file could be keyed on social security number, with the name as a data field. All these actions are controlled by the contents of the dictionary for the new file. The steps in this process are:

- Use RMSNEW to create a file of the same length as XXX, call it YYY.
- Create the dictionary for YYY with the desired format.
- Copy the data from XXX.RMS to YYY.RMS using:

```
OS9: rmscopy xxx yyy
```

- YYY could now be renamed to XXX if desired.

8.3. To merge two files

Assuming XXX and YYY have the same dictionary formats, then to combine all the records from both files into YYY enter the command:

```
OS9: rmscopy xxx yyy
```

8.4. To post a transaction file to a master file

This operation is the same as a merge. As an example of this use, assume that an accounting function is to be performed with RMS. There may be a MASTER file of accounting transactions which is to contain only permanent date. There may also be a BATCH file which has the same record format as the MASTER file but is much shorter. Each time a new set of transactions are to be entered, for example each week they are entered into the file BATCH.RMS using the RMS editor. Then it is possible to run various validation printouts using the RMS REPORT program. These may include batch totals, or trial balance runs. If necessary, then the RMS editor is used to make corrections to the BATCH file until the reports balance properly. Then the RMSCOPY program can be used to “POST” the BATCH file to the MASTER file by merging it record-by-record. Once the is accomplished the BATCH file can be retained for archival, or deleted and re-created with RMSNEW.

```
OS9: rmscopy batch master
```

Chapter 9. Compatibility of RMS with other languages and utilities

RMS has been designed specifically for maximum compatibility with other forms of access. RMS files are, for the most part, simple text files, which can be read by most common utilities and by user-written BASIC09 programs. It is just as easy to read an RMS file with an assembler of Pascal program. Though it is a little more dangerous to do, it is also possible to update an RMS file in the same manner. The information in this section is not necessary in order to use RMS effectively. It is provided for the programmer who whishes to extend the power of RMS, or to use it as part of the solution to a larger problem.

9.1. Format of an RMS file

There are actually 4 types of files used by RMS. They are:

xxxxxx.RMS an RMS main data file

xxxxxx.DIC an RMS dictionary file

xxxxxx.REP a REPORT specification file

xxxxxx.NDX an INDEX file

The **.DIC** and **.REP** files are usually created by a text editor, and are by definition, text files. The **.NDX** file is normally created by the INDEX program, but its format is that of a text file. Without going into great detail, a text file has the following characteristics:

- Each line may be any length.
- Each line consists of printable ASCII characters only, that is hexadecimal $20 to $7E, or decimal 32 to 126.
- Each line is terminated by a RETURN character, hexadecimal $0D, or decimal 13. On disk, lines are stored contiguously; each line begins after the RETURN character of the previous line. This lines are packed together, crossing sector boundaries as necessary.

The RMS main data files are also text files, except that they have a few more characteristics:

- Every record (line) is the same length.
- The file is created all at once, and all records are written with fill characters. Later, data is written into the records.
- The first character of each record has a flag character in it.
- The first record has some other system information in it.

The first character of each record will have one of the four following flag values:

U This is an unused record; no data has been stored in it yet.

1 The record currently contains a primary record.

2 This record currently contains a secondary record.

D The record used to contain data but has been deleted; it may be subsequently re-used for a primary or secondary record.

When the RMSNEW utility is run, all records are filled with U. The first record in the file contains some vital information about the file. This record, which is marked with a “U”, will never be changed by RMS, and will never be used for other data. Changing this record can render the entire file garbage, so great care must be taken in user-written programs that are to write into the file. The specific data stored there may be read and used by programs. It consists of two binary numbers. The first is a 16 bit (2 byte) number. This is the length of each record in the file. Note that the length includes the flag byte, trailing RETURN, and all field data in between. These two values are filled in by the RMSNEW utility. The rest of the first record is reserved for future use by RMS.

Within the RMS file, all the other records are available for storage of primary or secondary records. The fields are arranged within the record in the same order are they are found in the dictionary. If the fields do not fill the allotted record length, then the unused characters will be blank in each record. The unused characters occur after the last field and before the RETURN.

Because of the method used to assign a location in the file to each new record that is written into it, the data records in the file are, in general, scattered pseudo-randomly throughout the entire file. This method provides for extremely fast access to any particular record with a minimum of disk overhead. The method, called Hash Coding, is described below.

9.2. Hash coding

Hash coding is simply a method of passing the key field through a mathematical function in such a way that the entire set of possible key values are mapped pseudo-randomly into the number of records allotted in the file. The same formula is always used, so the record can be found later by applying the same function to the key field again. Since it is possible for more than one different key value to be mapped to the same record location, hash coding requires that there be rules for resolving such problems in an orderly way. These rules are accomplished automatically by RMS. You need not be concerned with them unless you plan to have random access to the file from another language. In general terms:

To FIND a primary record, given its key value:

1. the key value is passed through the hash formula, this produces a number, n, between 1 and the file size,
2. read record n,
3. if the record has a “U” in column 1, then our record is NOT in the file.
4. if the record has a “1” in column 1, and it has the same value in the key field as the one we want, then we have found the desired record.
5. otherwise, the record is a deleted one, or a secondary, or a primary other than the one sought. In this case, if we have looked at more than 255 records so far, then give up, we will not find our record. Otherwise, add one to the record number, n. If n is greater than the size of the file, then set n to 1. Now go back to step 2 above.

To FIND a secondary record, given the location, n, of its associated primary record:

1. add one to n, if it is over the file size, then set n to 1.
2. read record n,
3. if it is a “U” record then there are no more secondary records in this group,
4. if it is a “2” record, and the key field matches that of the sought record group, then this is the next secondary of the group,
5. otherwise, if we have looked at over 255 records since the last record of this group, then give up, there are no more. If we have looked at fewer than 256 records, then go back to step 1.

To DELETE a record:

1. FIND the record, as shown above, and read it into memory.
2. change the first column of the record to “D”.
3. re-write the record to the same spot.
4. If the record was a primary record, and if secondary records exist in the file, then you must also delete all secondary records that belong to this primary record.

To MODIFY a record

1. FIND the record, as shown above, and read it into memory.
2. change any field EXCEPT the key field, never change a key field, or the record will never be found again.
3. re-write the record to the same spot.

To INSERT a new primary record:

1. Try to FIND the record as shown above; if we do, then we may NOT insert a record with the same key.
2. If a “D” record was passed along the way while we were searching for the record, then the new record is written into the spot occupied by the first “D” record passed.
3. If no “D” record was passed, and the search ended only after searching over 255 records, then we may NOT insert this record, there is no room.
4. If neither case 2 or 3 happen, then the search ended by running into a “U”; write the new record into that same spot.
5. The record MUST have a “1” in column 1, followed by the key value, and MUST end with a RETURN character. We may fill in any or all of the other fields. Fields are padded with spaces.

To INSERT a new secondary record:

1. FIND the associated primary record,
2. FIND the next secondary record in the group,
3. if a secondary record is found then go to step 2,
4. follow rules 2, 3 and 4 for inserting a primary record, to decide where to put this secondary record,
5. follow rule 5 for inserting a primary record, except that column 1 must contain a “2”.

9.3. About the hash coding algorithm

The particular hash coding formula used throughout all of the RMS utilities has been tested on large sets of various types of keys, including names, social security numbers, part numbers, etc. It works reasonably well on these, and other key sets that have some randomness in at least part of the key. If you happen to have a set of keys which seems to clump together rather than spread out in the file, then try adjusting the file size op to a larger prime number. This often helps spread the records more evenly in the file.

9.4. RMS and utilities

Since all RMS files are text files, then the following OS-9 utilities may be useful:

edit or stylograph	a text editor or word processor for creating and modifying dictionary and report spec files, and for entering index files.
LIST	to print out main data files, dictionaries, index files or report spec files.
COPY	to backup any RMS files.

Note that even though it is possible to modify a main RMS data file with a text editor, it is very dangerous. Changing the key field of a record will cause the hash coding algorithm to fail to find it. Also, changing the length of any record in the file will have the effect of making the remainder of the file totally inaccessible to RMS.

9.5. Access to RMS files from BASIC09

The two types of RMS files which are likely to be accessed from BASIC09 and other user-written programs are the main data files, and the index files. For this explanation, examples are given in BASIC09 format.

To create an index file which is to be used to drive the RMS editor or REPORT writer, you need only write the record key values to a sequential file. An example will demonstrate:

```
DIM PATH:INTEGER
CREATE #PATH,"abc.ndx":WRITE \(*create the index file
PRINT #PATH,"John Smith"     \(*write two key field values
PRINT #PATH,"Wilbur Wright"
CLOSE #PATH                  \(*close the index file
```

This example creates a file on the working drive. The file is named ABC.NDX, and it contains 2 lines. This file could be used by RMS as an index file.

The other type of RMS file that may be accessed is the RMS main data file. This file can be read as a sequential file, or it can be read and written as a random file. In either case, the records read in will all be the same length. They should be read into a string variable or BASIC09 user-defined structure. The first character will be the record type flag (U, D, 1 or 2). The rest of the string will be the data fields. This example shows how to read an RMS file sequentially.

```
DIM PATH:INTEGER
DIM RECORD:STRING[200] \(*assign a large enough string
DIM FLAG:STRING
OPEN #PATH,"CUSTOMER.RMS":READ \(*open the data file for read
REPEAT
  READ #PATH,RECORD \(*read a record into the string
  FLAG=LEFT$(RECORD,1) \(*get the flag character
  IF FLAG="U" THEN
  PRINT "UNUSED RECORD"
  ENDIF
  IF FLAG="D" THEN
  PRINT "  DELETED RECORD: ";RECORD
  ENDIF
  IF FLAG="1" THEN
  PRINT "  PRIMARY RECORD: ";RECORD
  ENDIF
  IF FLAG="2" THEN
  PRINT "SECONDARY RECORD: ";RECORD
  ENDIF
UNTIL EOF(#PATH) \(*until file exhausted
CLOSE #PATH
```

```
END
```

This example will read all records of the file. It will stop only when an end of file is reached.

To access the same file as a random file it must be treated as a series of fixed length records. To do so you must know the records size. The physical record size is 2 greater than the record size that was input when RMSNEW was run to create the file. For example, if the length put in was 61, then the actual physical record length is 63. The 61 is the number of bytes available for field data, the 2 extra bytes are for the flag byte at the beginning of the record and the RETURN at the end. You must also know the number of records in the file. This was also put in when RMSNEW was run. The number of records will always be an odd number. If you put in an even number, say 500, then RMSNEW will use the next odd number, 501.

Consider a simple example of stock records, for which the RMS dictionary is

```
"       STOCK RECORDS"
PART  6  B  "PART NUMBER: ";
NAME 20  A  "Part name: ";
LEVEL 5  N  "Stock level: ";
PRICE 7  M  "Item price: ";
$
```

Each record is 38 characters long. With the flag character and RETURN, the physical record length is 40 bytes. The following program reads the number or records and the record length from the “zeroth” record, and accesses the file by record number.

```
DIM PATH,NUMBER,LENGTH,TOTAL,STOCK:INTEGER
DIM RECORD:STRING[200]
DIM PLACE:REAL
DIM PRICE:REAL
OPEN #PATH,"STOCK.RMS":READ
GET #PATH,TOTAL \(*Read the number of records
GET #PATH,LENGTH \(*and the record length
INPUT "Which record shall I read? ",NUMBER
PLACE=NUMBER*LENGTH \(*calculate the record position in the file
SEEK #PATH,PLACE \(*seek to it
READ #PATH,RECORD \(*now read the record
(*Print the record
PRINT "Record is type ";LEFT$(RECORD,1)
PRINT "Part number is ";MID$(RECORD,2,6)
PRINT "Part name is ";MID$(RECORD,8,20)
PRINT "Current stock level is ";MID$(RECORD,28,5)
PRINT "Item price is ";MID$(RECORD,33,7)
PRICE=VAL(MID$(RECORD,33,7)) \(*get the item price
STOCK=VAL(MID$(RECORD,28,5)) \(*and the stock level
PRINT "Current stock value is ";PRICE*STOCK
CLOSE #PATH
END
```

This program has assumed that the record number you entered was valid primary record. Normally, RMS data files are accessed by the key field of the record. To do this we need the hash coding algorithm used by RMS. The algorithm used is simple and straightforward. The ASCII values of the characters in the key field are forced to upper case summated alternately into two counters, which are then combined to form the record number. An example will best show the details:

```
DIM TOTAL,NUMBER,M,N,MASK,I,C:INTEGER
DIM KEY:STRING[200]
INPUT "Enter key field: ",key
INPUT "Enter total number of records in the file: ",TOTAL
M=0 \N=0 \(*zero the counters
FOR I=1 TO LEN(KEY) STEP 2
  C=ASC(MID$(KEY,I,1))
  GOSUB 100 \(*force upper case
  M=M+C-$20 \(*subtract space character $20
  EXITIF I=LEN(KEY) THEN \(*exit loop if all chrs counted
  ENDEXIT
  C=ASC(MID$(KEY,I+1,1))
  GOSUB 100
  N=N+C-$20 \(*alternate chrs in M and N
NEXT I
N=MOD(N,256) \(*make N fit into a byte
MASK=1 \(*now make up a limiting mask
WHILE MASK <TOTAL/256 DO
  MASK=MASK*2
ENDWHILE
M=MOD(M,MASK) \(*mask the high byte
NUMBER=M*256+N \(*combine the two bytes into an integer
NUMBER=MOD(NUMBER,TOTAL) \(*ensure it is less than the file size
IF NUMBER=0 THEN
  NUMBER=1 \(*can't be 0
ENDIF
PRINT "Record number is ";NUMBER
END
100 (*convert to upper case
IF C>=ASC("a")AND C<=ASC("z")THEN
  C=C-ASC("a")+ASC("A")
ENDIF
RETURN
```

The above subroutine is coded for clarity rather than for minimum number of statements. It can, however be used as shown to compute the hash value for any key field in exactly the same way as the RMS programs do it.

Chapter 10. An example RMS application explained in detail

This short tutorial is intended to give you a quick and easy introduction to RMS. Follow it through, then go back and read the rest of the manual thoroughly, to pick up all the useful features of RMS. We suggest that you make frequent reference to the Key Assigment and Command Line summaries at the end of this manual.

To start with, make sure that you have made a backup copy of your RMS disk using the OS-9 '**backup**' command, and that you have put the original disk away in a safe place. Use the copy you have made for this exercise. Place your disk in drive 0, and set your execution directory (using 'chx') and your data directory (using 'chd'):

```
OS9: chd /d0/cmds chd /d0
```

If you are using a Dragon 64, and your screen is not already in the 51 characters by 24 lines mode, enter that mode by:

```
OS9: go51
```

You are now ready to run RMS.

This example is that of a simple inventory system. It consists of an inventory master file, keyed on part number; each record contains several pieces of information about each part. Though this example is a simple one, it shows how an application is put together with RMS since all the essential elements are present. The sample inventory file is contained on the RMS distribution disk as 2 files: SAMPLE.DIC and PRLIST.REP.

10.1. A sample dictionary - SAMPLE.DIC

The first step in implementing an application with RMS is to create a dictionary file. In this case the file is named SAMPLE.DIC; we have already created it for you. The root name (SAMPLE) must be the same as the name we will use for the master data file. The file type (.DIC) indicates that this is an RMS dictionary file. SAMPLE.DIC was created using a text editor; it can be listed by the **LIST** command. Such a listing is shown below.

```
OS9: list sample.dic

"RMS SAMPLE FILE: INVENTORY RECORD";
PARTNO   8   A   "PART NUMBER: " <6> ;
DESC     30  A   "DESCRIPTION: " ;
PRICE    7   M   "RETAIL PRICE: ";
INSTOCK  4   N   "NUMBER IN STOCK: ";
LOC      1   A   "STORED IN AISLE "  (A,F) ;
REORDER  3   N   "RE-ORDER WHEN STOCK DOWN TO " ;
COST     7   M   "WHOLESALE PRICE: ";
SUPPLIER 10  A   "SUPPLIED BY: ";
ONORDER  1   A   "BACK ORDERED NOW " [Y,N, ];
ESTDATE  8   D*  "SHOULD BE IN STOCK BY ";
```

The top line is the title line. The next 10 lines describe 10 fields within the record. The first is considered to be the key field. Down the left side are the field names: PARTNO, DESC, PRICE, INSTOCK, LOC, etc. These have been chosen to indicate what the field contains. The second column indicates the number of characters reserved in each record for the individual fields. Note that the total length is 79. This number is important when the master data file is created. The next column indicates

the type of data to be stored in each field. Note that all four types are used: Alphanumeric, Numeric, Money and Date. The asterisk, "*", after the D on the last line indicates that this field is optional. That is, it need not be filled in by the operator when storing a record. The next item on each line is a prompt. Note that each is enclosed in quotes. The prompt is an explanatory message to be built into the screen form as an aid to the operator who must fill out the form. Some of the prompts have extra spaces inserted at the beginning or elsewhere within the text. These spaces are placed in such a way as to arrange the screen form into a neat and pleasing display. This is done largely by trial and error. On the first try just put the prompts in with no extra spaces, as we have done. Then run the RMS editor to see what the form looks like. Then insert spaces or re-arrange the form by editing the dictionary. It will usually take just two or three attempts to get the desired display. After using RMS a few times you will gain a skill for this process.

Three of the fields have another item on the line. The three are PARTNO, LOC and ONORDER. The items are VALIDATORS. The purpose is to insure that the operator can not store a record in the file unless the data in these fields meet the given restrictions. The PARTNO field must have at least 6 non-blank characters in it. The LOC must be a letter greater than or equal to "A" and less than or equal to "F". In effect, LOC can be A, B, C, D, E or F. The ONORDER field can only be Y, N or blank.

Notice that each field spec line ends with a semi-colon. This is required by RMS.

The sample file will have primary records only. no secondary records. If secondary records were to be used, then a "$" would need to be typed after the last field spec line, then another record specification would follow.

10.2. A sample master file - SAMPLE.RMS

The master data file for the example will be called SAMPLE.RMS. Note that the root name "SAMPLE" matches that of the dictionary file and the file type is ".RMS" which indicates that this is an RMS master data file. You should create it by the command shown below using RMSNEW. RMSNEW will ask you for the record size and the number of records. Respond with 79 and 47 respectively.

```
        OS9: rmsnew sample

INPUT RECORD SIZE :
79
INPUT NUMBER OF RECORDS :
47
FORMAT IN PROGRESS...
FILE FORMATTED OK
```

This sequence need only be done once for any particular RMS file.The record size put in is 79, which will contain our record length, 79. The number of records, 47, was chosen arbitrarily. For an example, 47 is large enough to demonstrate how RMS works. The reason for 47 rather than 40 or 50 is that the hash coding scheme used internally by RMS is most efficient when the number of records in the file is a prime number. It is not absolutely necessary to use a prime, but it will increase the performance of RMS if you do.

Once formatted the file contains records filled with the letter "U". Normally a "U" in the first character or a record indicates to RMS that the record is so far unused. Thus all records in the file are unused. The first record of the file will have the "U" in the first character, then it will have some important overhead information in the nect few bytes. RMS will never write any other data in this record. Change the first record would have a very harmful effect on the validity of the file, so avoid doing so if you ever write any BASIC09 programs that access the file.

10.3. Using RMS on the sample file

The above two files, SAMPLE.RMS and SAMPLE.DIC are all that is necessary to try out RMS. To start the RMS editor enter:

```
OS9: rms sample
```

At present the data file is empty - all the records are “unused”. Try entering some records. Type in the characters that you want in each field. Use the BACKSPACE key to correct errors. Use the RETURN key to move to the next field in the record, and the LEFT-ARROW key to move to the previous field. When you are happy with the data, insert the record into the file with the INSERT key. Three “beeps” tells you that the record has been inserted. Then clear the form with the CLEAR key, ready to enter the next record. Enter about half a dozen records.

Now try FINDing and changing a record. Get an empty form with the CLEAR key. Type in the key field, (the Part Number in our example), or a record that you have previously inserted into the file. (If you can't remember any of the Part Numbers that you inserted, use the SCAN key to browse through the file). Now use the FIND key to read in the record from the data file. Go through and change any fields you like but NOT the key field, (the first field). You can change fields simply by overtyping the data in the field. Overtype unwanted characters with spaces. Nor use the UPDATE key to write the changed record back into the file. Three “beeps” tells you that the record has been successfully updated.

Lastly try DELETing a record. Display the record using FIND or SCAN. Press DELETE to delete the record. RMS will ask you if you are sure you want to delete the record. If you are sure, press D, otherwise press some other key. Don't delete all the records that you have entered - leave at least four, we will use the to demonstrate INDEX and REPORT.

To exit RMS and return to OS-9 use the quit key.

Note that when you push INSERT or UPDATE, RMS will check the validity of all fields. This consists of making sure all numbers are right justified, decimal points appear where they should, etc. Some corrections to the data will be made automatically and will appear on the screen. Also checked are the various validators for each field. If anything is wrong with the data a message is placed on the top line of the screen telling you what is wrong. You must then push a key to continue. If this happens the record is no stored into the master file. To do so you must correct the problem then try INSERT or UPDATE again.

You may also notice that while typing data into the form you hear a beep. This indicates that you have hit an invalid key. For example, a “G” would be invalid in a numeric field.

10.4. A sample index file - DESCRIP.NDX

The file DESCRIP.NDX is an index file. Its purpose is to determine the processing order of the records when printed by the sample report. (The sample report is described below). You should create this file by the command:

```
OS9: index sample descrip desc
```

“SAMPLE” indicates the name of the master data file, SAMPLE.RMS, and the dictionary SAMPLE.DIC. “DESCRIP” indicates that the index file to be created should be named DESCRIP.NDX. “DESC” is the name of one of the fields in the dictionary. Specifically, it is the name of the description field for each part number.

This command reads through the file SAMPLE.RMS and creates a list of PART NUMBERS, (the key fields,) in the file DESCRIP.NDX. The list will be in alphabetical order by the DESC (description) field associated with each PARTNE. After a few moments the INDEX program will indicate that it is done by the message: .NDX FILE CREATED OK.

If an old file named DESCRIP.NDX is found, it is deleted and replaced by the newly created one. The NDX file created by the above will be a text file, with one PARTNO per line. You can list it using the OS-9 '**list**' command:

```
OS9: list descrip.ndx
```

Since the NDX file is just a text file, there are other ways to create one. One such way is with a text editor or a word processor. To do so just type one part number per line. The part numbers need bnot be in any particular order, any order you want may be used. Also, it is not necessary to include all the part numbers, and any individual part number could be included more than once if desired.

Remember that an **.NDX** file is valid only until records have been inserted or deleted from the master file. Once the contents of the file are changed, the **.NDX** file must be recreated by entering the above command again. This can be done any time before the index file is utilized again by REPORT or RMS.

There may be as many index files as necessary associated with one RMS master data file. For example, one index may be in order by part number, another by supplier, another by aisle location, etc. Each **.NDX** file is created similarly, and may be given any root name you choose.

There are two places in which an index file can be put to use. One is in conjunction with a REPORT printout. This is described in more detail below. The second way is in conjunction with the RMS editor. To use the index file in this way enter:

```
OS9: rms sample descrip
```

This starts RMS with DESCRIP.NDX as the designated index file. The index file comes into play when the SCAN key is depressed. Note that each time it is pushed, another record is displayed. Note also that the records are selected and displayed in alphabetical order by the description field.

10.5. A sample report - PRLIST.REP

The file PRLIST.REP on the distribution disk is a report spec. It can be used to produce a price list with the description, part number and retail price of each item in the file.

PRLIST.REP was created using the OS-9 text editor. Since it is a standard text file it can be listed with the '**list**' command. For example:

```
OS9: list prlist.rep

L 1,1 ;
T ;
T "SAMPLE PRICE LIST IN ORDER BY PRODUCT DESCRIPTION"@8;
T ;
T "DESCRIPTION OF PRODUCT          PART NUMBER      RETAIL PRICE"@1 ;
T "_____"@1 ;
P DESC@1 PARTNO@35 PRICE@52 ;
X DESCRIP;
```

The first line "`L 1,1 ;`" indicates that the report is to be printed without any automatic skipping over page boundaries. The next 5 lines (starting with "`T`") are title page lines. Each indicates a line to be displayed at the top of the report printout. Two of these lines are "`T ;`". These indicate a blank line, for cosmetic purposes. The seventh line, starting with "`P`", indicates what is to be printed for each primary record in the file. In our case there are only primary records in the file, no secondary records. The "`DESC@1`" indicates that the field called DESC, the description field, should be printed starting in column 1 of the line. The "`PARTNO@35`" indicates that the part number field is to be printed starting in column 35. The price field is to be printed in column 52. The last line of the file PRLIST.REP indicates that an index file is to be used to determine the order that the record are to be printed. The

index file specified is DESCRIP.NDX. If the “X” command line were omitted from the **.REP** file, then the report would contain the same print lines, but they would not be in order by description. They would be in random order.

There may be any number of **.REP** files associated with one RMS master data file. Each one may have any desired root name and a file extension of “.REP”. For the sample inventory file it would be possible to produce many useful reports: a re-order list based on low stock; an inventory by supplier; a list of all back ordered items; a form for taking physical inventory which would contain a list of all parts, sorted by aisle, with the number the computer thinks are in stock and a blank spot to fill in the actual number in stock. Hopefully, you get the idea.

To print the sample report enter the following command:

```
OS9: report sample prlist
```

“SAMPLE” specifies that the main data file is SAMPLE.RMS and the dictionary is SAMPLE.DIC. “PRLIST” indicates that the report spec is found in the file PRLIST.REP.

The above command should produce output that looks something like this:

```
        SAMPLE PRICE LIST IN ORDER BY PRODUCT DESCRIPTION

DESCRIPTION OF PRODUCT          PART NUMBER      RETAIL PRICE
-------------------------------------------------------------
10 FT RUBBER GAS LINE             222222222           8.99
ADULT LIFE JACKET, LARGE          444444444          49.95
ADULT LIFE JACKET, MEDIUM         555555555          49.95
ADULT LIFE JACKET, SMALL          666666666          49.95
CHILDS LIFE JACKET, LARGE         999999999          13.95
CHILDS LIFE JACKET, MEDIUM        888888888          13.95
CHILDS LIFE JACKET, SMALL         777777777          13.95
PLASTIC BATTERY CADDY, 6 VOLT     333333333           5.00
PORTABLE 6 GALLON GAS CAN         111111111          47.50
```

Of course, what is actually displayed will depend on what you entered!

To send this report to a printer on the parallel port “/p” you would use the OS-9 redirection facility:

```
OS9: report sample prlist >/p
```

10.6. Some comments

Once you have seen how it is done, you should begin to get ideas about possible applications you have that can be done completely, or in part, by RMS. To start up an RMS application you need only create the dictionary, then run RMSNEW to create and format the master data file. Various report spec files can be added as they are needed. Any time a sorted index is needed, the INDEX utility can be used to create it with one simple command. Once the files are created, the user need only know how to use the RMS editor, and know the schedule for entering the commands that create index files and reports.

As with any computerized system, make sure that backup disks are made, and keps on a regular rotational schedule. This will guard against hardware malfunction, as well as operator errors.

We hope that RMS will become your “best friend”.

Appendix A. OS-9 Command line summary

A summary of OS-9 command line syntax for calling the RMS programs.

Command line	Produces

`OS9: stylo datafile.dic`
`OS9: edit datafile.dic`
stylograph word processor or OS-9 text editor used to create the dictionary that defines the format of the datafile.
	datafile.dic

`OS9: rmsnew datafile` Creates and formats a new RMS data file as specified by the user.	datafile.rms
`OS9: rms datafile` Calls up the RMS editor to allow the operator to inspect, modify or add to the data in the file.	
`OS9: rms datafile indexfile` Specifies an index file to be used in conjunction with the RMS SCAN command for an orderly scan of the data records.	
`OS9: index datafile indexfile sortfiled` Used to create an index file “indexfile.ndx” containing the key fields of the data file “datafile” sorted by field “sortfield”.	indexfile.ndx

`OS9: stylo reportspec.rep`
`OS9: edit reportspec.rep`
stylograph word processor or OS-9 text editor used to create the report specification needed by the REPORT program.
	reportspec.rep

`OS9: report datafile reportspec >/device` The REPORT program produces a report out to device “device” of the data in file “datafile.rms” in a manner defined by the report specification “reportspec.rep”.	
`OS9: report datafile reportspec >outfile` The same, except that the output goes to a disk file “outfile”.	outfile
`OS9: rmscopy source destination` Copies the records in “source.rms” to the file “destination.rms”. Can be used to merge files, change the size of the file, or change the record format of a file.	

Appendix B. Key assignments for the Dragon 64

On the Dragon 64 the CLEAR key is used as the CONTROL key. For example, where CONTROL-U is mentioned, this is achieved by holding down the CLEAR key and hitting the U key.

Key name	Dragon 64 key	Function
RETURN	ENTER	Move to next field
BACKSPACE	←	Delete a character
RIGHT-ARROW	→	Same as RETURN
LEFT-ARROW	SHIFT-←	Move to previous field
QUIT	CONTROL-E	Return to OS-9
CLEAR	SHIFT-↑	Clear the form
FIND	CONTROL-F	Find a record by key field
DELETE	CONTROL-D	Delete the displayed record
UPDATE	CONTROL-U	Update the record on disk
FEED	↓	Display the next secondary record
INSERT	SHIFT-→	Insert a new record on disk
HOME		Go to first (key) field
SCAN	SHIFT-↓	Browse through file
DUPLICATE	CONTROL-↓	Copy field from last record
[	CONTROL-8	
]	CONTROL-9	
CAPS-LOCK	CONTROL-0	