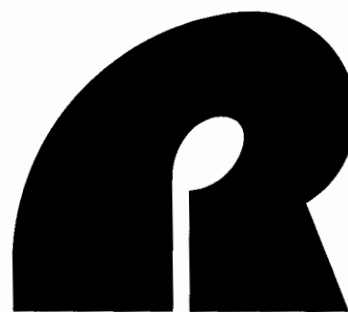


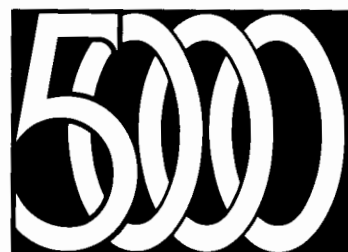


# USER'S MANUAL

R:BASE  
SERIES 5000  
PC-DOS



B A S E



Copyright © 1985. All rights reserved.  
By Microrim, Inc., Bellevue, Washington

## CREDITS

Concept and Direction	Wayne Erickson, David Kroenke
Documentation	Marva Dasef, Eric Getsinger, Kenneth A. Requa, Mark Williams
Programming and Design	J. Keith Bankston, Fred Gray, Charles N. Kawasaki, Alan C. Lindsay, Colin Miller, Greg Shumate, Tim Winston
Quality Assurance	Roger Dobratz, Leslie S. Gardenswartz, Mike Johnson, Kenneth C. Swenson, Katharine Warfield
Production	Cliff Hewitt, Ken Suvada
Product Marketing	Debra Gray, Krista Hansen, Gail Robbins, John Vanderwall
Corporate Dedication to Quality	All Employees of Microrim

## COPYRIGHT

COPYRIGHT© by Microrim, Inc., 1985. All rights reserved. No part of this publication may be reproduced, transmitted, transcribed, stored in a retrieval system, or translated into any language in any form by any means, without the written permission of Microrim, Inc.

## TRADEMARK

Microrim® is a registered trademark of Microrim, Inc.  
R:BASE SERIES™ is a trademark of Microrim, Inc.  
CLOUT and XRW are trademarks of Microrim, Inc.

AT™ is a trademark of International Business Machines Corporation.  
dBASE II™ and dBASE III™ are trademarks of Ashton-Tate.  
Diablo® is a registered trademark of XEROX Corporation.  
DIF® is a registered trademark of Software Arts Products Corporation.  
Epson® is a registered trademark of Epson America, Inc.  
IBM® is a registered trademark of International Business Machines Corporation.  
Lotus 1-2-3™ and Symphony™ are registered trademarks of Lotus Development Corporation.  
Multiplan® is a registered trademark of Microsoft Corporation.  
pfs® :FILE is a trademark of Software Publishing Corporation.  
VisiCalc® is a trademark of VisiCorp.  
WordStar™ and MailMerge™ are trademarks of Micropro International Corporation.  
XT™ is a trademark of International Business Machines Corporation.

## SOFTWARE COPYRIGHT NOTICE

Your license agreement with Microrim, Inc., authorizes the number of copies which can be made and the computer system(s) on which they may be used. Any unauthorized duplication or use of R:base 5000 in whole or in part, in print or in any other storage-and-retrieval system, is forbidden.

## DISCLAIMER

Names and references to persons, corporations, or products that are used in the examples of the *R:base Series 5000 User's Manual* and *Reference Manual* are intentionally fictional. Any resemblance to persons living or dead or to actual corporations or products is purely coincidental.



Second Edition  
R:BASE SERIES™ 5000  
User's Manual  
UD5013-01 Version 1.01  
September 1985

**HP Computer Museum**  
**[www.hpmuseum.net](http://www.hpmuseum.net)**

**For research and education purposes only.**

## **PREFACE**

### **R:BASE 5000**

R:base 5000 is the state-of-the-art solution to database development and management. The complete procedural language enables you to create powerful applications, and the visual display lets you do it without entering line after line of code. If you have not used a product like this before, you can request help and R:base responds with on-line instructions and screens that look like the database you are creating. If, however, you are an experienced DBMS user or application developer, R:base lets you get right to work with its powerful command language.

R:base 5000 consists of four major parts: R:base commands, the Application EXPRESS, the FileGateway, and RCOMPILE. You can use R:base commands to define or redefine databases and the information within them, assign security passwords, define data entry rules, create custom forms and reports, and build command files. The Application EXPRESS is a breakthrough in database and application development. When you use the EXPRESS, you are guided through each task with screens that prompt you for information and show you what your final product looks like. After you finish using the EXPRESS, it automatically creates the internal code using R:base commands. The FileGateway is a data-transfer package designed to format information created by popular programs such as Lotus 1-2-3, dBASE II, and PFS:FILE. You then can use R:base relational commands to organize the information into a meaningful report or screen display. RCOMPILE is the R:base application compiler. After you finish designing an application, use RCOMPILE to convert the ASCII file you created to a binary file. The binary file protects your program from changes and reduces the execution time.



## HOW TO USE THIS MANUAL

The *R:base 5000 User's Manual* has three parts.

Part 1, *Getting Started with R:base 5000*, tells you what you need to know to put R:base 5000 to work for you. The first chapter describes how to install and start up the product. Chapter 2 introduces database management and how to do it with R:base 5000. Chapters 3 and 4 are tutorials that help you learn R:base and discover how easy it is to build an application system for your database with the Application EXPRESS.

Part 2, *Managing a Database*, provides nine chapters with detailed information on all aspects of the R:base database management system and the FileGateway. When you first use R:base, at least skim through these chapters to find the information you need immediately. Then come back when you want to know more and read in detail. Chapters that describe individual commands are arranged so you can read only the section of the chapter that describes the command for which you need immediate information. Chapters that describe processes, such as creating a report, are arranged so that you can follow all necessary steps in proper order.

Part 3, *Building Applications*, has four chapters that show how to build an application system. Chapter 14 shows how to use the Application EXPRESS for quickly producing a menu-driven application system. Chapter 15 provides detailed information on the R:base 5000 programming language that allows you to customize your R:base applications. Chapter 16 describes RCOMPILE. Chapter 17 provides an overview of a sample application built with R:base 5000. The working application is included on one of your R:base 5000 product disks.

For quickly finding the information you need in this manual, use either the main table of contents that follows this preface or the detailed table of contents at the beginning of the chapter you are using. This manual also has a detailed index and glossary.

## R:BASE 5000 REFERENCE MANUAL

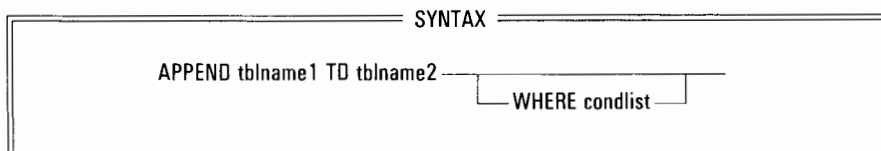
The *R:base 5000 Reference Manual* is designed to provide quick assistance on R:base commands, Application EXPRESS, FileGateway, RCOMPILE, and all R:base 5000 error messages. When you are working with R:base 5000, keep the *Reference Manual* close at hand.

## A NOTE ON R:BASE COMMAND FORMAT

Throughout this manual, commands and keywords—the words you enter to instruct R:base—are printed in upper case and arguments—the conditions and requirements you include for a command—are printed in lower case.

COMMAND KEYWORD argument KEYWORD argument

R:base syntax (the structure of a command) is displayed in this manual, in the *R:base 5000 Reference Manual*, and on R:base 5000 help screens in a format like this:



The syntax is read from left to right with no breaks in the flow of the main line of the syntax. Optional portions of a command are offset from the command line.

In the example, the word APPEND, on the main line, is required. The following words—*tblname1 TO tblname2*—are also required. The word group *WHERE condlist* (a list of conditions) is optional and, if omitted, does not break the left-right flow of the syntax main line.

The keyword-argument combinations vary with each application of this command. In the example shown above, the COMMAND is APPEND, the KEYWORDS are TO and WHERE, and the ARGUMENTS are *tblname1*, *tblname2*, and *condlist*. The table names, column names, and conditions, which are provided by the user, must be of the type shown in the syntax. For example, *tblname* indicates entry of a table name. The R:base reserved words (commands and keywords) do not change.

For further information on R:base commands, see chapter 5.

## **R:BASE 5000 USER'S MANUAL CONTENTS**

### **Part 1. Getting Started with R:base 5000**

Chapter 1	
INSTALLING AND STARTING R:BASE	1-1
Chapter 2	
INTRODUCTION TO DATABASE MANAGEMENT WITH R:BASE 5000	2-1
Chapter 3	
INTRODUCTORY TUTORIAL	3-1
Chapter 4	
ADVANCED TUTORIAL	4-1

### **Part 2. Managing a Database**

Chapter 5	
R:BASE FUNDAMENTALS	5-1
Chapter 6	
DATABASE STRUCTURE	6-1
Chapter 7	
DATA ENTRY	7-1
Chapter 8	
DATA INQUIRY	8-1
Chapter 9	
DATA MODIFICATION	9-1
Chapter 10	
RELATIONAL OPERATIONS	10-1
Chapter 11	
REPORTS	11-1
Chapter 12	
DATA TRANSFERS	12-1
Chapter 13	
FILE GATEWAY	13-1

**Part 3. Building Applications**

Chapter 14  
THE APPLICATION EXPRESS ..... 14-1

Chapter 15  
THE R:BASE 5000 PROGRAMMING LANGUAGE ..... 15-1

Chapter 16  
RCOMPILE ..... 16-1

Chapter 17  
SAMPLE APPLICATION ..... 17-1

**Index With Glossary ..... I-1**





## Installing and Starting R:base 5000 Contents

<b>How to Use This Chapter</b>	1-2
<b>System Requirements</b>	1-2
<b>Recommended Option</b>	1-2
<b>Installing and Starting R:base 5000</b>	1-3
Hard Disk Systems	1-3
System Configuration	1-3
Installation	1-4
Path Command	1-5
Using 320K or More RAM	1-5
Using 256K RAM	1-6
Floppy Disk Systems	1-7
Preparation of Disks	1-7
System Configuration	1-7
Start Up	1-8
Preliminary Steps	1-8
R:base	1-9
Application EXPRESS	1-9
FileGateway	1-9
RBEDIT	1-10
RCOMPILE	1-10
<b>Using R:base 5000 with Your Printer</b>	1-10
<b>Customizing the Main Menu</b>	1-11
Calling the <i>Rb5000.cmd</i> File	1-12
Color Display	1-12
Adding a Program Name to the Menu	1-13
Storing the Edited File	1-14
<b>R:base Color Display</b>	1-14
Hard Disk Systems	1-14
Floppy Disk Systems	1-15
<b>Additional Buffers</b>	1-16
<b>Convert Utility for R:base 4000 Rules, Forms, and Reports</b>	1-16
<b>R:base 5000 Database Specifications</b>	1-18
<b>Disk Contents</b>	1-19

## HOW TO USE THIS CHAPTER

This chapter describes the system requirements for R:base 5000. It gives you instructions for installing the R:base 5000 software on your system, configuring your system, and starting R:base 5000. It also includes information on the following:

- Options for customizing your R:base main menu, for providing a color display for R:base, and for adding buffers
- R:base 5000 database specifications
- The contents of the R:base 5000 product disks

## SYSTEM REQUIREMENTS

R:base 5000 is designed to run on the IBM PC, PC-XT, PC-AT, and 100% compatible microcomputers.

Your system must have the following:

- DOS, version 2.0 or higher.
  - 237K of main memory available after system configuration. A minimum of 320K is recommended.
  - A color or monochrome monitor
  - A hard disk drive and one double-sided, double-density 5.25-inch floppy disk drive  
or  
Two double-sided, double-density 5.25-inch floppy-disk drives.
- A hard disk system is recommended.

## RECOMMENDED OPTION

For the full use of R:base 5000, you should have a printer. R:base 5000 can use most printers that are compatible with the computers and operating system listed above, and it can prepare reports up to 131 columns wide.

## INSTALLING AND STARTING R:BASE 5000

### Hard Disk Systems

#### SYSTEM CONFIGURATION

To accommodate R:base 5000, your system needs a system-configuration file (*config.sys*) with this command:

`FILES=20`

To alter the file, use the R:base 5000 configuration program. This program is on R:base disk I.

1. Insert R:base disk I in drive *a:*. Make *a:* the default drive.
2. At the *A>* prompt, type *config* and press [ENTER].

The program displays this message:

```
This program alters your system configuration file (CONFIG.SYS)
to accommodate R:base. You must complete this step and reboot your
system to make R:base run properly.
```

```
Does your system boot from a floppy disk (y/n)
```

3. Answer the questions for your system. The program then creates or alters the *config.sys* file so that it will prepare your system for R:base each time you boot.

When finished, the program displays this message:

```
Configuration completed.
```

```
You must reboot your system (Ctrl-Alt-Del) before the
new configuration will take effect.
```

4. Remove the R:base disk from drive *a:*. Then reboot your system. Hold down [Ctrl] and [Alt] and then press [Del].



**INSTALLATION**

Microrim, Inc., authorizes the licensee to make two copies of each R:base 5000 disk. Make one copy on your hard disk and one on floppy disks as a back-up copy.

Throughout these instructions, it is assumed your hard drive is *c:* and the directory to which you copy the files is *r5k*. If this is not the case, substitute the correct drive designator or directory name.

A hard disk gives you options for how to arrange R:base files, your database files, and your application files. R:base 5000 files do not need to be on the same directory as database and application files. One of several convenient arrangements is the following (see figure 1-1).

- Put the R:base 5000 files on a directory (*c:\r5k*).
- When you create your database and application, place them on another directory (*c:\dbdir*).
- Define a path to the directory containing the R:base 5000 files. See “Path Command” below.

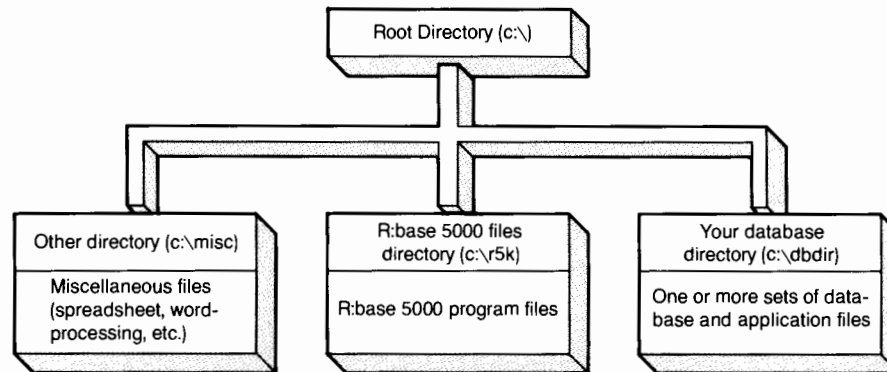


Figure 1-1 R:base Files and Database Files on Different Directories

To install the R:base 5000 files, follow these steps.

1. Boot up your computer and use the DOS CHDIR (change directory) command to make your current directory the one on which you want the R:base 5000 files to reside—*c:\r5k*.

If necessary, make the directory using the DOS MKDIR command. At the C> prompt, type:

```
MKDIR c:\r5k
```

2. Insert one of the R:base 5000 disks in floppy drive *a*.
3. Copy all files from the disk to your current directory. At the C> prompt, enter:  

```
COPY a:*,*
```
4. When all files from the disk are copied, remove the disk from drive *a*.
5. Repeat steps 2 through 4 until you have copied all R:base 5000 files to your hard-drive directory.

When you have copied all files to your hard disk directory, you are ready to use the full R:base 5000 product.

Store your R:base 5000 product disks in a safe place.

#### **PATH COMMAND**

If you will be using R:base 5000 from a directory other than the directory which holds the R:base 5000 files, you need to issue this DOS command at the C> prompt:

```
PATH c:\r5k
```

If you choose, you can add the command to your DOS automatic execution batch file, *autoexec.bat*. Then DOS will execute the command each time you boot your system. Enter the command line in the batch file, like this:

```
PATH c:\r5k
```

See your DOS manual for more information on editing and using the *autoexec.bat* file.

#### **USING 320K OR MORE RAM**

1. Boot up your system with DOS.
2. Change from the root directory to the directory where your database files are located. Enter the DOS CHDIR (change directory) command like this:

```
CHDIR c:\dbdir
```

3. Set a path to the directory with the R:base 5000 files (as described above in "PATH Command"). At the C> prompt, enter:

```
PATH c:\r5k
```

4. At the DOS prompt, type *RB5000* and press [ENTER].

When the R:base 5000 main menu appears, you can begin using R:base 5000.

Move the cursor between program names on the menu by using the left-arrow or right-arrow key or the tab key. When the cursor is over the name of the program you want to run, press [ENTER].

Your options are:

- EXPRESS: the Application EXPRESS
- GATEWAY: the FileGateway file-conversion program
- RBASE: the R:base database management system
- RBEDIT: the file editor
- RCOMPILE: the application program compiler
- CLOUT: The natural language inquiry program (not supplied with R:base 5000)

### **USING 256K RAM**

1. Boot up your system with DOS.
2. Change from the root directory to the directory where your database files are located. Enter the DOS CHDIR (change directory) command like this:

```
CHDIR c:\r5k
```

3. Set a path to the directory with the R:base 5000 files (as described above in "PATH Command"). At the C> prompt, enter:

```
PATH c:\r5k
```

4. At the DOS prompt, enter the name of the program you want to run.

Your options are:

- EXPRESS: the Application EXPRESS
- GATEWAY: the FileGateway file-conversion program
- RBASE: the R:base database management system
- RBEDIT: the file editor
- RCOMPILE: the application program compiler
- CLOUT: The natural language inquiry program (not supplied with R:base 5000)

After you enter the name of your selection and press [ENTER], the main menu for the program will appear. You can now begin using the program.

## Floppy Disk Systems

### PREPARATION OF DISKS

Microrim, Inc., authorizes the licensee to make two copies of each R:base 5000 disk: one as your working copy and one as a master copy.

To make copies, follow these steps:

1. Insert one of the R:base 5000 disks in floppy drive *a*: .
2. Insert a blank, formatted floppy disk in drive *b*: .
3. Copy all files from the R:base disk to your blank disk. At the A > prompt, enter:  

```
COPY *.* b:
```
4. When all files are copied, remove both disks.
5. Repeat steps 1 through 4 until you have made copies of all R:base 5000 disks.

If you choose now to make a second set of disk for your master copies, repeat steps 1 through 4.

Do not place write-protect labels on your working copies. Use the working copies when you use R:base 5000. Store your R:base 5000 product disks and master disks in a safe place.

### SYSTEM CONFIGURATION

To accommodate R:base 5000, your system needs a system-configuration (*config.sys*) file with this command:

```
FILES=20
```

To alter the file, use the R:base 5000 configuration program. This program is on R:base disk I.

1. Boot up your system with DOS.
2. Insert R:base disk I in drive *a*: .

3. At the A> prompt type *config* and press [ENTER].

The program displays this message:

```
This program alters your system configuration file (CONFIG.SYS)
to accommodate R:base. You must complete this step and reboot your
system to make R:base run properly.
```

```
Does your system boot from a floppy disk (y/n)
```

4. Press [Y] for yes and then press [ENTER].
5. As directed, remove the R:base disk and insert the boot disk in drive *a:* .

The program then creates or alters the *config.sys* file on your boot disk so that it will prepare your system for R:base 5000 each time you boot.

When finished, the program displays this message:

```
Configuration completed.
```

```
You must reboot your system (Ctrl-Alt-Del) before the
new configuration will take effect.
```

6. Reboot your system. With the boot disk in drive *a:* , hold down [Ctrl] and [Alt] and then press [Del].

## **START UP**

### **Preliminary Steps**

1. Boot your system with DOS.
2. At the A> prompt, enter this DOS PATH command:  
PATH a:\  
Press [ENTER].
3. Remove the boot disk from drive *a:* .

4. Make *b:* the default drive. At the A> prompt, enter: *b:* .

You are now ready to insert the disk or disks for the R:base 5000 program you want to use.

To eliminate step 2 above, you may want to make a special boot disk for R:base 5000. Include on the disk an automatic execution batch file (*autoexec.bat*) with this command:

```
PATH a:\
```

Then, whenever you boot your system, DOS will execute the command. See the DOS manual for information on editing and using the *autoexec.bat* file.

### Application EXPRESS

1. Insert in drive *a:* your working copy of the EXPRESS program disk.
2. Insert in drive *b:* a blank formatted disk or your database application disk.
3. At the B> prompt, type *express* and press [ENTER].

When the main menu appears, you can begin using the EXPRESS.

### FileGateway

1. Insert in drive *a:* your working copy of the FileGateway program disk.
2. Insert in drive *b:* the disk that has the files to be converted to R:base format.
3. At the B> prompt, type *gateway* and press [ENTER].

When the main menu appears, you can begin using FileGateway.

### R:base

If you want R:base on-line help available, copy the file named *help.rbs* to the disk you will use for your database. If you want R:base prompts available, copy the file named *prompt.rbs* to the disk you will use for your database. *Help.rbs* and *prompt.rbs* are located on the R:base 5000 utilities disk. Neither file is required for the operation of R:base.

1. Insert your working copy of disk I in drive *a:* .
2. Insert a blank, formatted floppy disk (or your database disk) into drive *b:* .
3. At the B> prompt, type *rbase* and press [ENTER].
4. When R:base prompts you to change disks, remove disk I from drive *a:* and insert your working copy of disk II into drive *a:* . Press any key.

When the R:base main menu appears, you can begin using the R:base database management system.

### **RBEDIT**

1. Insert in drive *a*: your working copy of the RBEDIT program disk.
2. Insert in drive *b*: either a blank, formatted disk on which you will create and store new files or a disk with the files you will edit.
3. At the B> prompt, type *rbedit* and press [ENTER].

When the main menu appears, you can begin using RBEDIT.

### **RCOMPILE**

1. Insert in drive *a*: your working copy of the RCOMPILE program disk.
2. Insert in drive *b*: the disk with the files that you are going to compile.
3. At the B> prompt, type *rcompile* and press [ENTER].

When the main menu appears, you can begin using RCOMPILE.

## **USING R:BASE 5000 WITH YOUR PRINTER**

R:base 5000 is designed to operate with any printer you can attach to your IBM PC or compatible microcomputer. If your printer responds to the [Shift][PrtSc] or the [Ctrl][PrtSc] keys, then R:base can use your printer with any function which allows output to a printer.

If you have a dot matrix printer, such as an Epson, you have available a number of options which enable you to produce attractive report formats. These options include but are not limited to:

- Compressed print
- Boldface type
- Expanded print
- Varied line spacing

These, and other, printing options are invoked by definition in a report format. Chapter 11 explains how to enter printer control codes in your report format.

A letter-quality printer, such as a Diablo, does not allow the variations on print options that a dot matrix printer allows. You may be able to set the pitch using switches on the printer itself. For example, you can print a report which is up to 120 characters wide on 8.5-inch wide paper by setting the pitch to 15. Check your printer manual for information on adjustments you can make to a letter-quality printer.

Another possible method of adjustment is to use the DOS MODE command to change the number of characters printed per line (80 or 132) or the number of lines printed per vertical inch (6 or 8). The MODE command is entered at the DOS prompt:

```
MODE LPT#:   n     ,m     ,P  
```

where # is 1, 2, or 3 for the printer port number, *n* is 80 or 132 to set the characters per line, *m* is 6 or 8 for the number of lines per vertical inch, and *P* indicates continuous retry on time-out errors.

If you have more than one printer attached to your computer, you will also need to use the MODE command at the operating system level to switch from one printer port to the other. See your DOS manual for details.

## CUSTOMIZING THE MAIN MENU

If you have a color monitor, you can customize the main menu command file, so that the main menu is displayed in the colors you specify. For a color display, you name the foreground and background colors in the *rb5000.dat* file.

If you have Microrim's Extended Report Writer (XRW), you can add its name to the R:base main menu. You can then call it in the same way you call the R:base 5000 programs. To be able to call a program from the R:base 5000 main menu, you need to do two preparatory procedures:

- Copy the program files to the same directory as your R:base 5000 files. (See your product installation and start-up instructions for exceptions.)
- Add the name of the program to the *rb5000.dat* file.

The *rb5000.dat* is an ASCII file that contains:

- The R:base SET COLOR commands for foreground and background colors
- A list of program names

You can use any ASCII editor—RBEDIT, DOS EDLIN, or any other you are accustomed to using.

The following directions show you how to use RBEDIT to edit the file. The examples use red and blue as the colors and XRW as the product to be added. If you are adding other colors, substitute those names. (See chapter 5 for a list of available colors.) If you are adding another Microrim product, substitute its name.



## Calling the *Rb5000.dat* File

1. Start up RBEDIT. The main menu appears.

```
-----Old file--New file--Quit-----R:base screen editor
```

2. Press [O] for *Old file*.
3. When RBEDIT asks for the file name, enter:

```
rb5000.dat
```

This screen is displayed:

```

                                     < 1, 1> [ESC] to exit
SET TEXT "Select an option or press [ESC] to return to DOS"
SET COLOR FORE GRAY
SET COLOR BACK BLACK
EXPRESS -R
GATEWAY -R
RBASE -R
RBEDIT -R
RCOMPILE -R
CLOUT2
```

## Color Display

To change the colors from gray foreground and black background to red foreground and blue background, you edit the second and third commands.

1. Use the cursor-control keys to place the cursor on the second line. Change the foreground-color command to read:

```
SET COLOR FORE RED
```

You do not need to retype the entire command. Move the cursor down to the word *GRAY* and delete it with the [Del] key or by typing over it.

2. Place the cursor on the third command and change the background-color command to read:

```
SET COLOR BACK BLUE
```

With your entry in place, the screen should look similar to this:

```

                                                    < 1, 1> [ESC] to exit
SET TEXT "Select an option or press [ESC] to return to DOS"
SET COLOR FORE RED
SET COLOR BACK BLUE
EXPRESS -R
GATEWAY -R
RBASE -R
RBEDIT -R
RCOMPILE -R
CLOUT2

```

3. If you are also going to add a program name to the menu, read "Adding a Program Name to the Menu."

If you are finished, press [ESC] and read "Storing the Edited File."

### Adding a Program Name to the Menu

1. Place the cursor on the line below the last name on the list.
2. Add the name you use when you start up the product at the DOS prompt. For the Extended Report Writer, you enter *XRW*.
3. Do not include *-R* with your entry. The *-R* shown after each R:base 5000 program indicates that the large R:base *R* is not to be displayed.

With your entry in place, the screen should look similar to this:

```

                                                    < 1, 1> [ESC] to exit
SET TEXT "Select an option or press [ESC] to return to DOS"
SET COLOR FORE RED
SET COLOR BACK BLUE
EXPRESS -R
GATEWAY -R
RBASE -R
RBEDIT -R
RCOMPILE -R
CLOUT2
XRW

```

4. Press [ESC] to exit.

## Storing the Edited File

1. When you press [ESC], RBEDIT displays this menu:

```
---Edit again---Save file---Next file---Quit-----R:base screen editor
```

Press [S] to save the file.

2. RBEDIT needs to know the name under which you will save the file. Since it is *rb5000.dat*, the name that appears in brackets, press [ENTER] to keep the name.

3. You are finished, so press [Q] to quit.

Your customized R:base 5000 main menu is now ready for use.

## R:BASE COLOR DISPLAY

If you have a color monitor, you can set R:base and RBEDIT, when called from within R:base, to display in color. The EXPRESS, FileGateway, RBEDIT, when called from the R:base 5000 main menu or from the DOS prompt, and RCOMPILE do not display in color.

To display R:base in color, you can use the SET COLOR command after you start R:base. If you use a hard disk system, however, you will probably find it more convenient either to make an *rbase.dat* file or to edit an existing *rbase.dat* file so that it contains the commands to set the foreground and background colors you want and brings up R:base in color automatically. When R:base starts up, it first looks for an *rbase.dat* file on the default drive and current directory. If R:base finds the file, it executes the commands in it.

If you use a two floppy disk system, you can make a small command file that you can run within R:base.

## Hard Disk Systems

To enter the color commands in the *rbase.dat* file, use DOS EDLIN, RBEDIT, or the editor of your choice. The following directions show you how to use RBEDIT to edit the file. The examples use red and blue as the colors. If you are adding other colors, substitute those names. (See chapter 5 for a list of available colors.)

1. Start up RBEDIT. The main menu appears.

```
--Old file--New file--Quit-----R:base screen editor
```

2. Press [O] for *Old file* if you already have an *rbase.dat* file, or press [N] for *New file* if you are going to create an *rbase.dat* file.
3. When RBEDIT asks for the file name, enter:  
`rbase.dat`
4. Type the SET COLOR command for the foreground and background, like this:  
`SET COLOR FORE red`  
`SET COLOR BACK blue`
5. When you press [ESC], RBEDIT displays this menu:

```
---Edit again---Save file---Next file---Quit-----R:base screen editor
```

Press [S] to save the file.

6. RBEDIT needs to know the name under which you will save the file. Name it *rbase.dat*.

Remember that the file must be on the same drive and directory as the database with which it will be used. If you are not on that drive and in that directory, add the appropriate drive designator and directory name.

When you have entered the name, drive designator, and directory, if needed, press [ENTER].

7. You are finished, so press [Q] to quit.

Your *rbase.dat* file is now ready to start R:base in color.

## Floppy Disk Systems

If you use a two floppy-disk system, you cannot use an *rbase.dat* file, but you can achieve similar results by making a command file that will hold the commands to set the foreground and background colors.

To create the file, follow the steps outlined in "Hard Disk Systems." Give the file a name unique in its directory (but not *rbase.dat*), such as *rbcolor.dat*. Place the file on the directory with the database on default drive *b:*.

To run the file, start up R:base, and from the main menu select [1], the command mode. At the R> prompt, enter the RUN command with your file name, like this:

```
RUN rbcolor.dat
```

Press [ENTER] to execute the file and to set the screen to the colors you have specified in the commands contained in the file.

## ADDITIONAL BUFFERS

DOS uses buffers to hold data being read from or written to a disk. The DOS default is two. If you have available RAM memory after you load R:base, you have the option of improving performance by using the memory for additional buffers. By increasing the number of buffers, you allow DOS to allocate buffer space to more files. Then, when DOS needs to read or write a file, a buffer is already available for use. This increases the speed of read and write operations. DOS, however, searches the buffers for the correct file. Therefore, if you allocate too many buffers, the processing speed can actually decrease since DOS is forced to search through empty buffers.

Here is a summary of steps to follow to check for available memory and to add buffers.

1. Boot up your system and start up R:base.
2. At the R> prompt, enter the CHKDSK command. R:base shows you how many bytes of memory are still available. Each buffer uses 528 bytes. A good range for increased performance is between 10 and 20.
3. To add buffers, edit your system configuration file (*config.sys*). You can use DOS EDLIN, RBEDIT, or the editor of your choice. (See "Customizing the Main Menu" and "R:base Color Display" in this chapter for examples of how to use RBEDIT.)

The number of buffers you add depends on available memory. Suppose you want to make 16 buffers available. Add this command to *config.sys*:

```
BUFFERS=16
```

4. Store the edited *config.sys* file.
  5. Reboot your system by pressing [Ctrl], [Alt], and [Del] simultaneously.
- Your system is now ready.

## CONVERT UTILITY FOR R:BASE 4000 RULES, FORMS, AND REPORTS

If you are a user of R:base 4000, you need to convert R:base 4000 databases with rules, forms, and reports, so you can use them with R:base 5000. This section shows you how to use CONVERT, an R:base command file, to accomplish the conversion. CONVERT accomplishes the following:

- It modifies the FORMS table
- It creates an R:base 5000 REPORTS table from an R:base 4000 REPORTER table
- It creates an R:base 5000 RULES table from an R:base 4000 RBSRULES table

Before using CONVERT, check the old REPORTER table for eight-character variable names. To do this, at the R> prompt, type:

```
EDIT rdata FROM reporter
```

Look at the variable names and, if they are eight characters long, insert a space to separate the name from the expression operand. In addition, make sure that none of the variable names is an R:base 5000 reserved word. (For a list of reserved words, see chapter 5.) If you have used an R:base 5000 reserved word as a variable name, change the variable name in both the variable definition and layout sections of the report format (look for the words VARIABLES and LAYOUT in the report format).

To convert your database, follow these steps:

1. Start up R:base 5000.
2. At the R> prompt, enter:

```
RUN convert.cmd
```

Include the drive and directory, if different from the default drive and directory. The CONVERT menu is displayed.

3. When prompted, enter the name of your database. If it is not on your default drive or current directory, include the drive and directory. Press [ENTER].

CONVERT begins to run, displaying this message: *Working...*

When the CONVERT program finishes, you see an R> prompt on the screen. The FORMS and RULES tables are now ready for use with R:base 5000.

If your database has reports, you will see a message informing you of the additional step you need to perform if your report has variables. Follow these steps:

1. At the R> prompt, type *REPORTS*.
2. Enter the name of your report.
3. From the REPORTS main menu, choose *Define*.
4. R:base lists the variables from your report. Check the data type of each variable. If the data type is incorrect or the word *ERROR* appears, you must reenter the expression.
5. Press [ESC] to exit from Define.
6. From the main menu, select *Quit*.
7. To store your converted variables, choose *Save changes*.

R:base will store your variables in the REPORTS table.

You can now use your R:base report.

Repeat the procedure for each R:base 4000 report with variables.

When you are satisfied that the new RULES and REPORTS tables are correct, you can remove the old RBSRULES and REPORTER tables from the database.

## **R:BASE 5000 DATABASE SPECIFICATIONS**

### **Maximums:**

- Number of tables per database — 40
- Number of columns per database — 400
- Size of row — 1530 characters
- Number of rows per table — Limited to maximum DOS file size
- Rows per database — Limited to maximum DOS file size
- Command line input — 1500 characters

### **Data Types:**

- DATE—Represented as MM/DD/YY or in any order you specify
- TIME—HH:MM:SS representing hours, minutes, and seconds
- DOLLAR—Range of  $\pm \$99,999,999,999.99$
- INTEGER—Range of  $\pm 999,999,999$
- REAL—Range of  $\pm 9 \times 10^{\pm 37}$  with 6-digit accuracy in scientific or decimal notation
- TEXT—one to 1500 characters

### **Data Storage for files established per database:**

- File 1 (directory)—9600 bytes (fixed).
- File 2 (data)—six bytes per row for internal pointers plus data fields. DATE, REAL, TIME, and INTEGER require four bytes each. DOLLAR requires eight bytes. TEXT is the column width, plus one byte if width is odd, with minimum storage being four bytes.
- File 3 (key index)—Approximately 12 bytes per key value.

### **File Compatibility:**

Databases created using R:base 4000 and CLOUT are fully compatible with R:base 5000.

## **DISK CONTENTS**

### **R:base Disk I**

RBASE.EXE  
INITIAL.RBS  
CONFIG.COM  
RBASEI.ID

### **R:base Disk II**

RBASE.OVL  
MESSAGE.RBS  
RBASEII.ID

### **Application EXPRESS Disk**

EXPRESS.EXE  
EXPRESS.HLP  
EXPRESS.MSG  
EXPRESS.ID

### **FileGateway Disk**

GATEWAY.EXE  
GATEWAY.DAT  
ASCII.DEL  
ASCII.FIX  
DBASEII.DBF  
LOTUS.WKS  
MULTIPLA.SYL  
PFSFILE  
VISICALC.DIF  
GATEWAY.ID

### **Utilities Disk**

PROMGEN.EXE  
RCOMPIL.E.EXE  
RBEDIT.EXE  
RB5000.EXE  
CLOUT2.EXE  
HELP.RBS  
PROMPT.RBS  
PROMPT.DAT  
RCOMPIL.MSG  
RBEDIT.MSG  
COMPUCO1.RBS  
COMPUCO2.RBS  
COMPUCO3.RBS  
CONVERT.CMD  
RB5000.DAT  
MACRO.DOC  
BACK.MAC  
DELFR.MAC  
FREQ.MAC  
LABELS.MAC  
POST.MAC  
SUMM.MAC  
UTILITY.ID

### **Sample Applications**

MENUS.ORD  
PROCS.ORD  
ORDER1.RBS  
ORDER2.RBS  
ORDER3.RBS  
PROCS1.ASC  
PROCS2.ASC  
PROCS3.ASC  
MENUS.ASC  
SAMPLE.ID





---

## Introduction to Database Management with R:base 5000

<b>How to Use This Chapter</b>	2-2
<b>Information Management with R:base 5000</b>	2-2
<b>Designing a Database</b>	2-3
Designing a Data Model	2-3
Characteristics of Tables Used for Data Models	2-5
Steps in Database Design	2-6
List the Basic Kinds of Information You Want	2-7
Decide Which Facts Are Important	2-7
Represent the Relationships	2-8
Improving the Design	2-8
List the Basic Kinds of Information You Want	2-9
Decide What Facts are Important	2-9
Adding a Table for Product Components	2-10
Represent the Relationships	2-10
Recording One-to-Many Relationships	2-15
Choosing the Links	2-15
Recording Many-to-Many Relationships	2-17
The Complete Design	2-17
<b>Implementing a Database with R:base 5000</b>	2-18
Defining the Database Structure	2-19
Loading the Database with Data	2-20
Manipulating the Database to Answer Questions	2-21
Manipulating One Table	2-21
Manipulating Multiple Tables	2-22
<b>Applications with R:base 5000</b>	2-24
Application Operator, Owner, and Developer	2-24
Application Design with the Application EXPRESS	2-27
Designing Menus	2-28
<b>The Next Steps</b>	2-31

## HOW TO USE THIS CHAPTER

This chapter shows you how to design databases that use the full power of R:base 5000 to serve your needs for information. It also shows how you as a database designer can use R:base 5000 to build database applications for the needs of other users.

It is recommended that you read this chapter before you begin using the product or the tutorial. This chapter is intended primarily for readers not familiar with databases or building database applications. If you have developed databases and database applications with R:base 4000 or other products, you may want to skim, watching for new features, tips, and details you can use to make your job easier and more efficient.

## INFORMATION MANAGEMENT WITH R:BASE 5000

A database helps manage data and produce information. It provides a way of recording and storing data that you want to keep and to have access to. Names, addresses, numbers, and dates are typical of the kinds of items that make up the data you need to store—and to have at hand for quick answers to questions that arise.

R:base 5000 is a relational database product that lets you design a set of tables to store and to retrieve your data easily. An R:base 5000 database is a collection of up to 40 of these tables. Each R:base database may have up to a combined total of 400 different columns for all tables. The columns can have different data types to fit all your needs. For example, in a column you can store a dollar number as high as \$99,999,999,999.99 or a piece of text, such as a memo, as large as 1,500 characters.

The power of R:base is in the control it gives you over the tables you create. You can manipulate the data from one or any combination of the tables in your database. For example, a sales representative can use R:base to store customer information in one table, sales information in another, and product information in a third. Tables can be organized so that any particular piece of information will only rarely need to be entered more than once. R:base lets you put together information from the tables so that the combined information answers the questions you ask.

## DESIGNING A DATABASE

R:base gives you the tools to build a database that will serve your unique needs for information. To make your database perform as well as possible, however, plan ahead. A good start is sketching out on paper a model of your database. Refine the model. Then, from your plan, use R:base to build the database on your computer.

This introduction provides an example of how to design a particular database. Although it is merely an example of one kind of database and thus provides only a partial picture, it does illustrate important concepts to remember when you set out to design your database.

### Designing a Data Model

When you maintain information, you are using a model that helps you keep track of the information you need. This model must be dynamic so it can change as your needs change. Consider an example. Compatible Computer Company (COMPUCO) makes and sells computers. The sales manager, June Wilson, sets out to design a database to answer questions for the Sales department. Of the people, events, and things she deals with, these categories are evident:

- The computer products that COMPUCO sells
- The customers who buy the products
- The sales representatives who sell the products

Most questions that come up are answered with information from one or more of these categories. To keep informed about them, the Sales department builds a data model. Figure 2-1 shows a simple data model with a table for each category. Individual sets of related data are represented by rows in the tables, and the facts that the Sales Department is going to keep about the objects are represented by the columns of the tables.

Sales Representative		
First Name	Last Name	Date Hired
Peter	Coffin	11/26/83
Ernest	Hernandez	08/28/82
Mary	Simpson	06/01/84
John	Smith	08/15/85

Sales Transaction				
Sales Rep Last Name	Customer Company Name	Computer Name	Units Sold	Date Sold
Coffin	Computer Distributors, Inc.	Portable Standard PC	10	03/12/85
Hernandez	Industrial Computers, Inc.	Advanced Color PC	8	03/12/85
Simpson	Industrial Concepts, Inc.	Classic Color PC	10	03/13/85
Smith	Computer Mountain, Inc.	Portable Standard PC	25	03/13/85

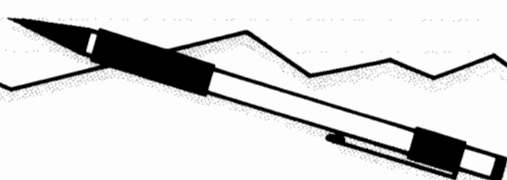


Figure 2-1 COMPUCO Data Model

Figure 2-1 shows that there are four sales representatives. The facts that are kept in the *sales representative* table are *first name*, *last name*, and *date hired*. Information about sales transactions is kept in another table. The facts that are maintained about sales transactions in that table include: *sales rep last name*, *customer company name*, *computer name*, *units sold*, and *date sold*.

Even with this simple model, the sales manager can answer a number of questions about sales transactions. For example:

- Who are the COMPUCO sales representatives?
- What sales did COMPUCO make in early March, 1985?
- Which sales representative has been with COMPUCO the longest?
- Which sales representative has sold the most portable PCs?
- How many Advanced PCs have been sold?
- What was the last purchase by Industrial Computers?
- Does Industrial Concepts, Inc., buy the standard portable PC?

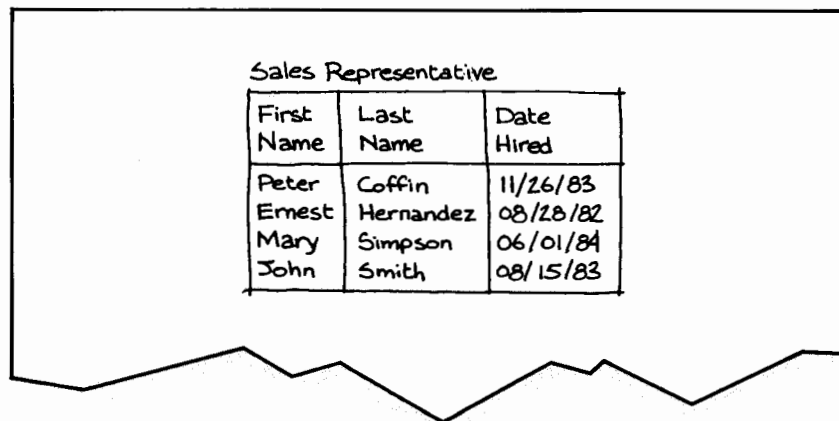
Some questions can be answered by looking at only one table, because the table refers to only one kind of information. Others require looking at both tables, because the kinds of information that the tables represent are related to one another. The model represents these relationships in terms of the data values which the tables have in common. In the example in figure 2-1, the common data which links the *sales representative* table to the *transaction* table is the last name of the sales representative.

Good models do not just happen. The more carefully you design, the closer the result will fit your needs. Set out to make an accurate model. That way you can get information to answer questions that arise.

## Characteristics of Tables Used for Data Models

The table that serves for the data model in figure 2-2 displays several characteristics.

- Every row has the same number of facts in it: *first name*, *last name*, and *date hired*.
- Each column contains the same kind of fact in each row. For example, *date hired* for the Hernandez row conveys the same fact about Hernandez as *date hired* conveys about Coffin, Simpson, or Smith.
- There is only one entry for each fact. That is, there can be only one date hired. If one of the sales representatives quits and is subsequently rehired, a policy decision will have to be made. Does the sales manager store the original or most recent date as the date hired? Or does she add another column to the table to store another fact about the sales representatives: *rehire date*? The answer depends upon what information the sales manager wants from the data model.



First Name	Last Name	Date Hired
Peter	Coffin	11/26/83
Ernest	Hernandez	08/28/82
Mary	Simpson	06/01/84
John	Smith	08/15/83

Figure 2-2 The *Sales Representative* Table

In this example, no two rows are exactly the same. Even if the company had two sales representatives with the same name, the rows would still be distinguished by the facts in the other columns. This characteristic of the table implies another characteristic: the order of the rows is unimportant. To get information about John Smith, you do not have to know the location of his row in the table. All you need to know is a fact or combination of facts about him which will identify his row. You can identify his row if you know his last name or hire date, since no other employee has these facts in common. A table like this also has another characteristic: the order of the columns is not important. In this case, *date hired* can as well go before *first name* and *last name*.

An effective data model contains the right kinds of facts and the right relationships among the various kinds of data. Models are approximations of what they represent. They contain only as much detail as they need to serve their purpose. The goal of data model design is to include enough detail so that the only question which cannot be answered is the question that is never asked.

## Steps in Database Design

To see the steps involved in creating a solid database design, consider again the example of the COMPUCO Sales department. The sales manager needs to know the answers to questions such as:

- What kinds of computers has the company sold?
- How many are available for immediate sales?
- Who are the customers?
- How many computers has each sales representative sold?
- What were the total sales for January?

From the first, consider what kinds of data you want to store and what kinds of information you want to get from your database. The three basic steps in designing a database are:

- List the basic kinds of information you want
- Decide what facts are important
- Represent the relationships

## LIST THE BASIC KINDS OF INFORMATION YOU WANT

Sales manager June Wilson needs information about sales transactions, computers sold, customers who bought the computers, and sales representatives who sold the computers to the customers. The list looks like this: *transaction*, *product*, *customer*, *sales representative*. For each, she will have a table.

## DECIDE WHICH FACTS ARE IMPORTANT

Next, decide what facts are needed to answer questions.

To answer questions about sales transactions, the sales manager needs to know which computers were sold, how many were sold, who sold them, who bought them, when they were bought, and what their selling price was.

To answer questions about the products that COMPUCO sells, the sales manager needs a product name to identify each product. The other piece of information she needs is the list price of the product.

Given the kinds of questions that the sales manager expects to answer, all that is needed at this point is the name of the customer, the customer's company, the company's address, and its phone number.

At this point, all the sales manager needs to know about the sales representative is a name.

Figure 2-3 lists the kinds of information the sales manager wants, placing the columns with the appropriate tables.

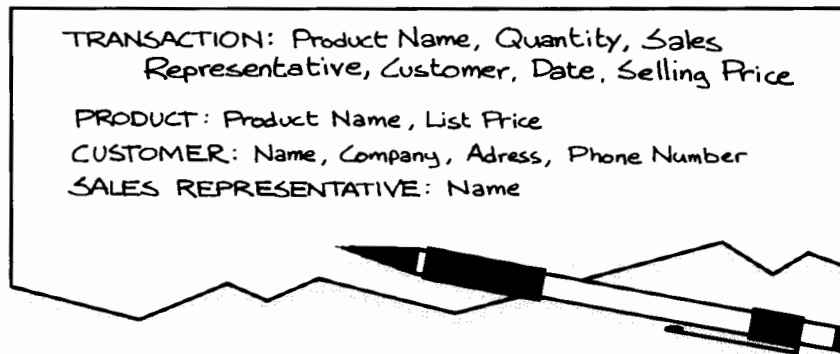


Figure 2-3 The COMPUCO Sales Manager's List



## REPRESENT THE RELATIONSHIPS

Be sure the database reflects the appropriate relationships among the various tables of names, things, and events. In the example, the way in which to proceed is to look at *transaction*, *product*, *customer*, and *sales representative* and make sure that the model represents the actual relationships.

What relationships do sales transactions involve? Sales are related to products. Their relationship can be described as one-to-many. That is, there can be many transactions for each product, but only one product is involved in each transaction. Since relationships between tables are represented by facts that they have in common, the link to products is represented by the product name.

There are also one-to-many relationships between the sale and the customer and between the sale and the sales representative. Each sale is to one customer, but a customer may make many purchases. Each sale is carried out by one sales representative, but a sales representative may make many sales.

Figure 2-4 shows these relationships by including the product name, the customer name, and the sales representative name in *transaction*.

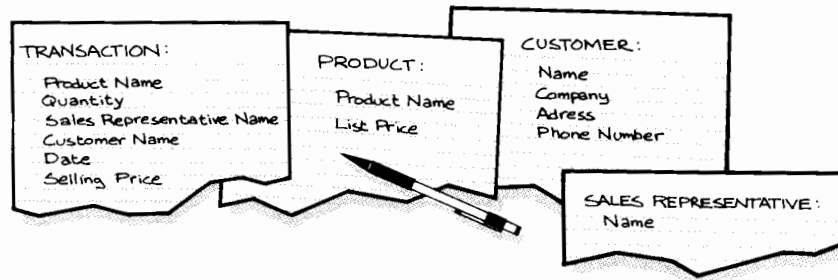


Figure 2-4 Relationships in the Data Model

## Improving the Design

Having built a basic model, try to find ways to improve it. The method for improving is the same as for developing the initial plan: go through the same three design steps; list the basic kinds of information you want; decide what facts are important; and represent the relationships among those objects.

## LIST THE BASIC KINDS OF INFORMATION YOU WANT

In the example of the COMPUCO Sales Department, sales manager June Wilson has determined that to answer questions about sales, she needs information about sales transactions, products, customers, and sales representatives. The chief task for improving the design is looking more closely at each and, where necessary, expanding.

## DECIDE WHAT FACTS ARE IMPORTANT

The basic principle involved is that a table should contain information about one category only. The *transaction* table, for instance, should contain only the information needed to report a sales transaction accurately. Consider the following details that were included in the table.

The name of the customer is a vital piece of information. It answers the question: Who made the purchase? To have a more complete record, the sales manager also needs to know the company the purchaser works for, its address, and telephone. Although the sales manager may need to know all this information, it does not need to be recorded with the transaction. In fact, it is better not to include anything more than is necessary. Repeating information leads to errors and, in particular, makes accurate and complete updating difficult. The solution is to have a separate table that contains a complete list of customers and customer information. All that is needed in the *transaction* table is a link that will identify the customer, the company, its address, and its telephone.

To answer more questions, two of these categories—customer name and address—are subdivided so that the data is more specific. The *customer* table now has these columns: *first name*, *last name*, *company*, *address*, *city*, *state*, *zip code*, *phone number*.

The table can now provide more information about the customers, and it can keep the information together where it can be easily updated.

For the *sales transaction* table, some way of identifying the person who made the sale is needed. The sales manager, however, needs other information about the sales force, so the *sales representative* table (the one side of the *sales representative-sales transaction* relationship) can be expanded to hold this kind of information about the sales representatives. This table now has these columns: *first name*, *last name*, *home address*, *city*, *state*, *zip code*, *home phone*, *work phone extension*, *date hired*.



Information from the *sales representative* table can answer questions such as: Where should the company send Simpson's surprise bonus check? How long has Hernandez worked for the company? Combined with the *transaction* table, it can provide data to answer a question such as: What have been Smith's average monthly sales since he started? In the last year? The basic question about the product that the sales transaction must answer is: What product was sold? Each product, therefore, must be uniquely identified. The product name and additional information about the product can be stored in the *product* table.

To answer questions about sales, the sales manager needs answers to questions like these: What does it cost the company to make this product? What is the list price? How many units of this product are in stock? To answer questions about the product itself, the sales manager needs to know a product number, a product name, and a description. The *product* table now includes these columns: *product cost*, *list price*, *quantity on hand*, *product number*, *product name*, *product description*.

### **Adding a Table for Product Components**

The sales manager decides she needs to know the major components included in each product. This is because the products she deals with, COMPUCO desk-top computers, have three major components: a monitor, a system unit, and a keyboard. For these products, the company offers a choice of two monitors, two keyboards, and six system units.

To keep the various configurations straight and to make maintaining the information easy, the sales manager adds a component table to include information about what components go into each of the products. The table has these columns: *product number*, *component number*, *component description*.

### **REPRESENT THE RELATIONSHIPS**

When models get complicated, use some method to show the relationships. One way is to map them out, representing each as a box. Spread the boxes out so you can connect them in a way that describes the relationships.

The only relationship involving customers is the one between customer and transaction. Drawing a line between boxes will show this relationship exists. Next, decide what type of relationship exists between the *customer* and *transaction* tables: Is it one-to-many or many-to-many? The answer comes from the nature of the transaction that the sales manager is describing and the part that the customer plays in it. Each customer can take part in many transactions. That is, each customer can make a number of purchases. Each transaction, however, has only one customer who makes the purchase. The relationship, therefore, is one-to-many: one customer to many transactions. Figure 2-5 shows a method for illustrating this one-to-many relationship. Note the single arrow close to the customer box, indicating that it is the one side of the relationship, and the double arrows close to the transaction box, indicating that this is the many side of the relationship. The importance of determining this relationship will soon become clear when the sales manager needs information from both tables.

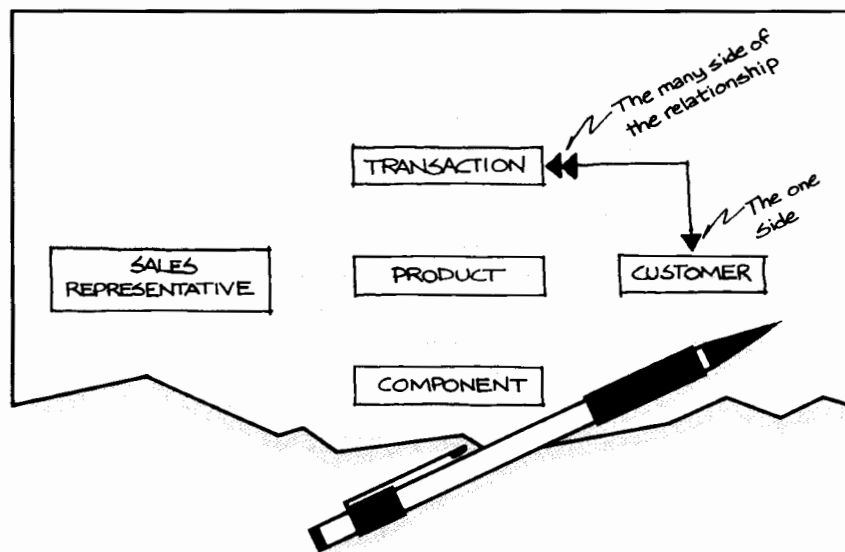


Figure 2-5 Illustrating the One Side and the Many Side

## 2-12 Getting Started with R:base 5000

Answers to questions often come from more than one table. For example, look at figure 2-6. If the sales manager needs to know the name of the person who purchased five Standard Color PCs on September 17, she gets the company name from the *transaction* table and the person's name from the company's row in the *customer* table. The question can be answered only if there is an entry in the *customer* table for Industrial Computers, Inc.

transaction					
DATE	CUSTOMER COMPANY	PRODUCT	UNITS	PRICE	SALES REPRESENTATIVE LASTNAME
09/17/84	Industrial Computers, Inc.	Standard Color PC	5	\$1,530	Simpson
09/24/84	Industrial Computers, Inc.	Advanced Color PC	5	\$2,575	Hernandez
10/10/84	Computer Distributors, Inc.	Standard PC	10	\$1,800	Smith
10/17/84	PC Distribution, Inc.	Portable Classic PC	8	\$3,000	Coffin

customer							
FIRST NAME	LAST NAME	CUSTOMER COMPANY	ADDRESS	CITY	STATE	ZIP	PHONE
Anna	Adams	Southwest Computers, Inc.	3015 55th NE	Austin	TX	78758	512-581-8800
Andy	Chin	PC Consultation and Design	7823 Foothills Road	Palo Alto	CA	94321	415-892-6745
Jane	Ferguson	Industrial Computers, Inc.	5200 Empire Way	Denver	CO	80214	303-239-7823
Walter	Finnegan	Computer Distributors, Inc.	24700 Industrial Parkway	Boston	MA	02162	617-423-8921
Sarah	James	PC Distribution, Inc.	3200 Westminster Way	Boston	MA	02178	617-341-2189
Betty	Jones	Midtown Computer Co.	2800 NE 47th	Chicago	IL	60637	312-277-5000
Bill	Jones	Computer Warehouse	14600 Eastgate Way	Bloomington	IN	47401	812-701-1002
Bill	Stevenson	Computer Mountain, Inc.	14792 15th Ave E.	Denver	CO	80214	303-271-1500
Shelley	Watts	Industrial Concepts, Inc.	5602 Silverdale Way	Livermore	CA	94550	415-878-5600

Figure 2-6 *Transaction and Customer Tables*

The model must include the requirement that no entry may be made for a given customer in the *transaction* table until the information for that customer has been entered in the *customer* table. Figure 2-7 shows how this requirement may be illustrated. The small line drawn across the relationship line near the *customer* box indicates that there must be an entry in the *customer* table before any entries for that item may be made in the *transaction* table. The small circle next to the *transaction* box indicates that there need not be an entry in the *transaction* table for every entry in the *customer* table. Notice that the optional side is the many side of the relationship and the required side is the one side. In this way, COMPUCO can record information for companies that have not yet made a purchase, such as sales leads or companies COMPUCO may want on an advertising mail list.

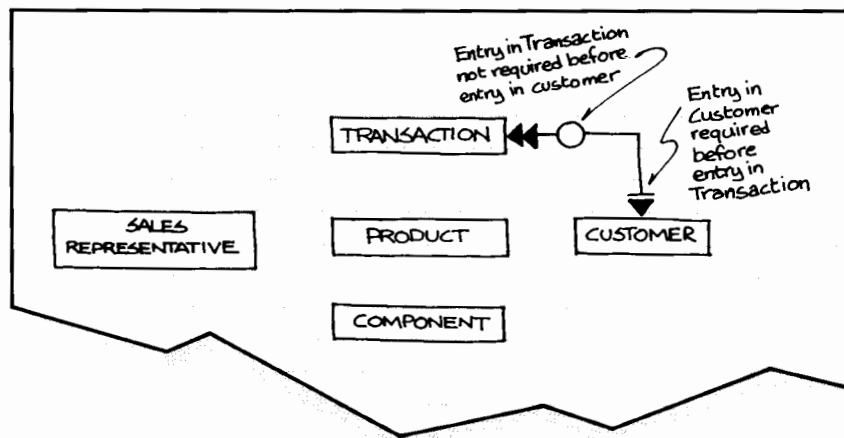


Figure 2-7 Requirements for the Transaction-Customer Relationship

The only direct relationship involving sales representatives is the one between sales representative and transaction. Each sales representative may have many sales, but each transaction involves only one sales representative. Thus, as was the case with the *customer* and *transaction* tables, the relationship is one to many: one sales representative to many transactions. The *sales representative* table, a complete list of all sales representatives, allows COMPUCO to record information about a sales representative whether or not that sales representative has made a sale. Before an entry is made in the *transaction* table for a sales representative, however, the relationship requires that an entry for that sales representative must appear in the *sales representative* table. This requirement is illustrated by placing a small line near the *sales representative* table, meaning that an entry for a sales representative must appear in this table before it may appear in the transaction table, and by placing a small circle near the *transaction* table, meaning that even if there is an entry in the *sales representative* table for a given sales representative, there need not be an entry for that sales representative in the *transaction* table.

The *product* table is related to the *transaction* table in a one-to-many relationship: each kind of computer can be included in many sales transactions, but each transaction involves only one kind of computer. Like the customer-transaction relationship and the sales representative-transaction relationship, this relationship will also require that no entry be made in the *transaction* table if the product information has not been entered in the *product* table. The sales manager can record information about products without making a corresponding entry for a sales transaction.

The *component* table is related to the *product* table in a many-to-many relationship: each component can be included in more than one computer, and each computer can have up to three components. Note that in figure 2-8 there are double arrows near both the product and component boxes in this relationship. The relationship will require that the product information not be placed in that table if the component information is not in the *component* table.

With all relationships between tables established, the database design now appears as shown in figure 2-8.

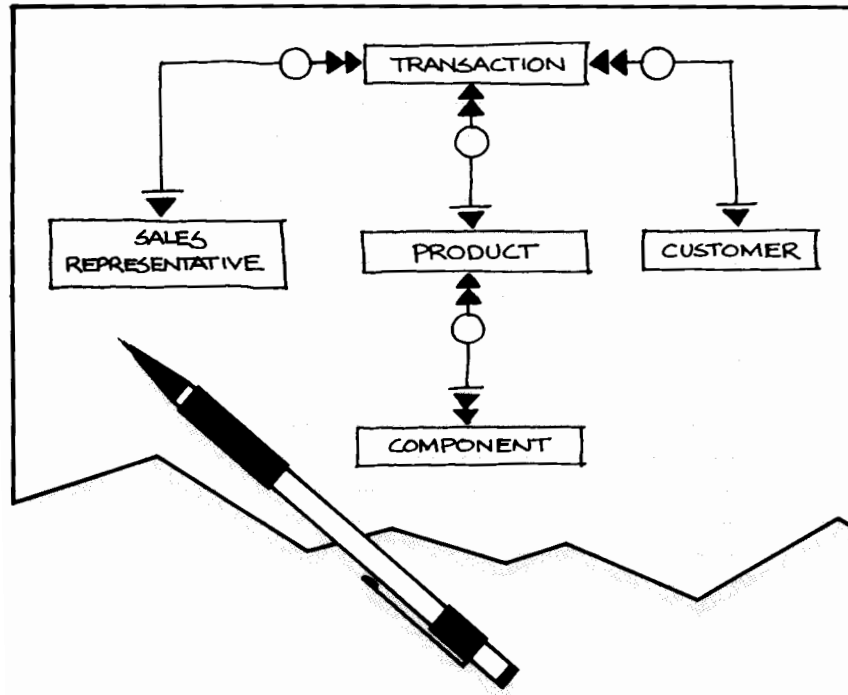


Figure 2-8 Relationships and Constraints: An Improved Design

## Recording One-to-Many Relationships

The final step in the design of the data model is to make sure that the proper links exist. In the COMPUCO example, three relationships are one-to-many: one sales representative, product, or customer to many transactions. One is many-to-many: many products to many components.

### CHOOSING THE LINKS

The column or columns you use to link two tables in a one-to-many relationship should be required to identify a row uniquely on the one side of that relationship. To see why, look again at the relationship between *customer* and *transaction*. Figure 2-9 shows sample rows from the tables using only *last name* as the link. Suppose the sales manager wants to find the telephone number for the customer who bought the Standard PC on November 9. To get the telephone number, she first looks in the *transaction* table to find the link.

transaction						
DATE	LAST NAME	PRODUCT	UNITS	PRICE	SALES REPRESENTATIVE	
10/17/84	James	Portable Classic PC	8	\$3,000	Smith	
11/09/84	Jones	Standard PC	15	\$1,800	Smith	

customer							
FIRST NAME	LAST NAME	COMPANY	ADDRESS	CITY	STATE	ZIP	PHONE
Bill	Jones	Computer Warehouse	14600 Eastgate Way	Bloomington	IN	47401	812-701-1002
Betty	Jones	Midtown Computer Co.	2800 N.E. 47th	Chicago	IL	60637	312-277-5000

Figure 2-9 Last Name as Link



Finding that the value of the link is *Jones*, she next looks in the *customer* table for the row with Jones in the *last name* column. Now she has a problem. The last name does not uniquely identify a row in the *customer* table. Which Jones is she looking for: Bill or Betty? To solve the ambiguity, she needs both the first name and the last name of the customer as the link. To serve as a link, the column or combination of columns must uniquely identify a row in the table that is being linked.

To avoid the problem of duplicate names, the link between *customer* and *transaction* needs to be either a combination of columns, such as *first name* and *last name*, or a unique column. A better choice than the name or names of the customer would be the name of the company. In the world of a business like COMPUCO, it would probably eliminate the need for combining columns to create a unique link. Another good choice is the telephone number. In this case, however, perhaps the best choice is a linking column called *customer number*. A customer number offers several advantages:

- It is unique. It eliminates any concern about customer or company names being identical to others and the need to use more than one column to identify a row.
- It is brief. In most cases, it will be briefer than a customer name, a company name, or a telephone number, such as 812-701-1002. On the many side of the relationship, where it is used often, it needs less space.
- It is easy to maintain. It makes updating a minimal task if, for example, Fred Harris replaces Bill Jones as the contact at Computer Warehouse, or if Midtown Computer Co. decides to change its name to MIDCOMPCO or MCC.
- Unlike text values, it is easier to enter exactly. For example, with text, you might remember a customer name as McMillan when it is really MacMillan.

A small company with a few employees may be able to use a last name as the linking column between the *sales representative* and the *transaction* tables. A large company, however, may run into the same kinds of problems with the names of sales representatives as it did with names of customers. Two or more employees may have the same name: John E. Smith and John L. Smith may both be sales representatives. An employee may change names: Mary Simpson may become Mary Thompson. As in the *customer* table, a better solution than using names which may not be unique is to use a unique number, such as an employee number or a social security number. For brevity (three digits instead of nine), COMPUCO uses an employee number. For the sales manager, this is a good choice for the linking column between the *sales representative* and the *transaction* tables.

For ease of entry and maintenance and for accuracy, a good choice for the link between *product* and *transaction* is a column for the product number. Like the customer number and employee number, it is unambiguous.

## Recording Many-to-Many Relationships

Although the column *component number* links the *product* table and the *component* table, the sales manager still has a data-entry and integrity problem. This is a many-to-many relationship. Each product uses many components, and each component is used in many products. In the current model, every time a different component is entered in the *product* table for a given product, all other information about that product must be entered again.

This problem can be solved by creating a linking table. Instead of including the component number in the *product* table, the sales manager creates a new table that includes two linking columns: one from the *product* table (*product number*) and one from the *component* table (*component number*). See figure 2-10. This linking table is named *components used*. The relationships in which *components used* participates are represented in figure 2-11. There is a one-to-many relationship between *product* and *components used* and a one-to-many relationship between *components* and *components used*. Since the sales manager does not expect duplicate entries for pairs of values in this table, no other columns are required to identify a row uniquely. Note the similarities in figure 2-10 between the *components used* table and the *transaction* table. The *transaction* table is actually a linking table too.

## The Complete Design

Figure 2-10 shows a list of tables and columns in the completed design, and figure 2-11 shows a map of the design.

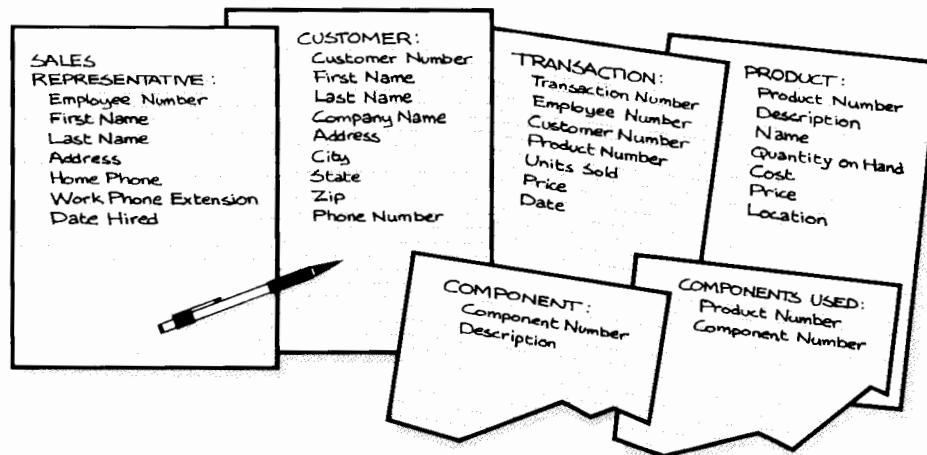


Figure 2-10 The Completed COMPUCO Data Design

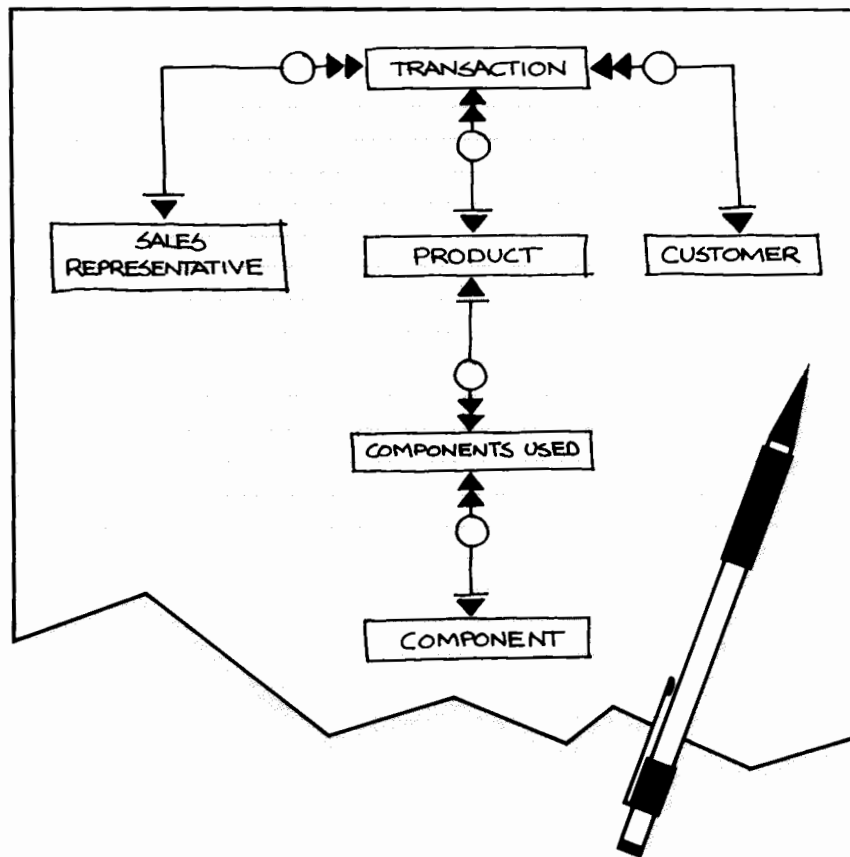


Figure 2-11 Relationships and Constraints: The Final Design

## IMPLEMENTING A DATABASE WITH R:BASE 5000

There are three steps to implementing your database design with R:base 5000:

- Define the database structure
- Load the database with data
- Manipulate the database to answer questions

The Tutorial provides detailed directions so that you can use R:base to create and to use a part of the sample COMPUCO database. The discussion that follows provides only a general overview of the implementation process, letting you watch over the shoulder of sales manager June Wilson.

## Defining the Database Structure

The database structure is what the data model represents. The task of implementation is bringing that model across into R:base. For defining a new database, there are two basic steps: naming the database and defining tables and their columns. Both steps are accomplished quickly and easily with R:base's Application EXPRESS.

Since R:base can build more than one database, the sales manager needs to have a unique name for each. The EXPRESS asks for a name for the database. The sample database is named *compuco*.

The EXPRESS then displays a table with seven columns. At the top left of the table is the box where the name of the table goes. At the top of each of the seven columns is a place to enter the name of the column. To define the table and its columns, the sales manager fills in the names in the appropriate boxes. Here is the *product* table with the table name and columns filled in.

Enter or change the column names—[ESC] when done

product						
prodid	proddesc	prodname	onhand	cost	listprce	location
INTEGER	TEXT 31	TEXT 35	INTEGER	DOLLAR	DOLLAR	

Select column data type - [ENTER] to choose

TEXT	DOLLAR	INTEGER	REAL	DATE	TIME
------	--------	---------	------	------	------

Database Compuco — Defining table Product — Column 7

Table and column names are eight or fewer characters, so some column names are abbreviated. For each entry of a column name, the EXPRESS has asked for a data type. The EXPRESS displays the choices below. The sales manager moves the highlighted block to the appropriate type to make the choice.

Choosing a data type means telling R:base what kind of entries will be made in the column. R:base provides six choices for your data. For example, if the column has alphanumeric data, such as names or addresses, choose TEXT. If the column has only whole numbers, choose INTEGER. When the choice is TEXT, the EXPRESS also asks for a length, the maximum number of characters that will be entered. Column seven, *location*, is a TEXT data type. When the [ENTER] key is pressed, the EXPRESS asks for the length and then places the entry in the data-type box below the column name.

The *product* table has only seven columns. For a table with more columns than that, the EXPRESS will shift the columns to the left as they are filled, leaving a column at the right for the next entry. The EXPRESS always keeps seven columns on screen and displays the number of the current column.

When all tables are defined, the definition of the database structure is complete.

## Loading the Database with Data

Data is loaded into the database one table at a time. R:base offers a number of ways to enter the data. One way is to design a form that will then appear on the screen so the user can fill in blanks. R:base stores the data in a table in the database. This method lets the sales manager design her own entry forms. For example, a screen entry form for customer information can look like this:

Push [ESC] when done with this data

### Customer Information

Customer Number:

First name:

Last name:

Company name:

Address:

City:

State: Zip code:

Phone number:

The sales manager can also enter data with the LOAD command. The prompt option with this command steps through the entry process by displaying the column name and its type as she enters the data. For a complete discussion on data-entry methods, see chapter 7.

## Manipulating the Database to Answer Questions

Once the data is entered, the sales manager is ready to begin manipulating the database to answer questions. Some questions can be answered with one table. Others require more than one. The following paragraphs provide only a quick overview of some available options. For a complete discussion, see chapter 8 and chapter 12.

### MANIPULATING ONE TABLE

The R:base SELECT command is used for making queries. Information can be selected from all columns in a table or from any one or more you specify. The sales manager can ask to see only the customer number, the last name of the customer, the company name, its city, and the phone number from *custlist* (the customer table). R:base responds with this screen:

```
R> SELECT custid lastname company city phone FROM custlist
```

custid	lastname	company	city	phone
100	James	PC Distribution Inc.	Boston	617-341-2189
101	Finnegan	Computer Distributors Inc.	Boston	617-423-8921
102	Ferguson	Industrial Computers Inc.	Denver	303-239-7823
103	Stevenson	Computer Mountain Inc.	Denver	303-271-1500
104	Watts	Industrial Concepts Inc.	Livermore	415-878-5600
105	Chin	PC Consultation and Design	Palo Alto	415-892-6745
106	Jones	Computer Warehouse	Bloomington	812-701-1002
107	Jones	Midtown Computer Co.	Chicago	312-277-5000
110	Adams	Southwest Computers Inc.	Austin	512-581-8800

Many questions require a narrower selection of data. For example, suppose the sales manager wants the telephone numbers of the customers in Boston. She can display the entire list of customers, as in the sample screen above, and look down the list to find the Boston companies. That screen, however, contains more data than she needs to answer her question.

She can receive a more specific answer by adding a limitation to her request: *WHERE city = Boston*. R:base responds with the information shown in the screen illustration below.

```
R> SELECT custid lastname company city phone FROM custlist WHERE city = Boston
```

<u>custid</u>	<u>lastname</u>	<u>company</u>	<u>city</u>	<u>phone</u>
100	James	PC Distribution Inc.	Boston	617-341-2189
101	Finnegan	Computer Distributors Inc.	Boston	617-423-8921

The sales manager can be even more demanding. For example, she may want to list customer information by company in alphabetical order. She tells R:base to sort the information the way she wants. R:base answers with the information shown in this screen illustration:

```
R> SELECT custid lastname company city phone FROM custlist SORTED BY company
```

<u>custid</u>	<u>lastname</u>	<u>company</u>	<u>city</u>	<u>phone</u>
101	Finnegan	Computer Distributors Inc.	Boston	617-423-8921
103	Stevenson	Computer Mountain Inc.	Denver	303-271-1500
106	Jones	Computer Warehouse	Bloomington	812-701-1002
102	Ferguson	Industrial Computers Inc.	Denver	303-239-7823
104	Watts	Industrial Concepts Inc.	Livermore	415-878-5600
107	Jones	Midtown Computer Co.	Chicago	312-277-5000
105	Chin	PC Consultation and Design	Palo Alto	415-892-6745
100	James	PC Distribution Inc.	Boston	617-341-2189
110	Adams	Southwest Computers Inc.	Austin	512-581-8800

## MANIPULATING MULTIPLE TABLES

You can answer a number of questions using only one table. If your tables, however, are well-constructed so that each deals with only one subject, you will need to retrieve information from more than one table to answer complex questions.

For example, in the COMPUCO sample database, *transx*, the transaction table, holds the keys to the other tables, but to get the details the sales manager needs information from those tables. Perhaps someone, who is not familiar with the customer identification numbers, asks her for information about which company is buying which computers. Before she can use the SELECT command, she needs to combine the *transx* and the *custlist* tables.

R:base provides four commands for handling combinations of tables. One that is useful in this case is the INTERSECT command. To combine tables, the sales manager types:

```
INTERSECT custlist WITH transx FORMING clsttrx
```

In carrying out this command, R:base builds a new table named, in this example, *clsttrx*—for *customer list* and *transaction*. This new table is made up of all columns from both *custlist* and *transx*. *Custid*, the link that was present in both tables, is presented only once. For the rows, R:base placed in the new table all rows from both *transx* and *custlist* where the rows have the same value in the column *custid*. Where it did not find a match, it did not bring over a row. For example, customer number 110, Southwest Computers is not included in this new table, because it has not purchased COMPUCO computers. Having combined the two tables, the sales manager can now use the SELECT command to answer the question. Here is the information for purchases on March 12, 1985:

```
R>SELECT tdate company prodid price FROM clsttrx SORTED BY company +
R>WHERE tdate = 3/12/85
```

tdate	company	prodid	price
03/12/85	Computer Distributors Inc.	MX3020	\$2,300.00
03/12/85	Computer Distributors Inc.	MX3030	\$2,575.00
03/12/85	Computer Distributors Inc.	PX3040	\$3,100.00
03/12/85	Computer Distributors Inc.	PB3050	\$2,700.00
03/12/85	Computer Mountain Inc.	PX3050	\$3,175.00
03/12/85	Computer Mountain Inc.	CX3000	\$1,800.00
03/12/85	Computer Mountain Inc.	PB3040	\$2,250.00
03/12/85	PC Consultation and Design	CX3020	\$2,575.00
03/12/85	PC Consultation and Design	CX3010	\$2,275.00
03/12/85	PC Consultation and Design	MB3000	\$1,600.00
03/12/85	PC Consultation and Design	MB3020	\$1,975.00
03/12/85	PC Consultation and Design	PX3020	\$3,100.00
03/12/85	PC Consultation and Design	CX3000	\$1,800.00
03/12/85	PC Distribution Inc.	CX3000	\$1,900.00



## **APPLICATIONS WITH R:BASE 5000**

R:base 5000 offers you the Application EXPRESS, and a full set of commands for building applications that make it easier for you or for other users to carry out database operations. An application includes an organized set of operator menus, informational screens, and commands that allow a user to perform functions such as adding rows, editing data values, and preparing reports by using screens that offer a predefined list of options. A user of an application does not need to know what the structure of your database is, how your application works, or how to compose a single R:base command. When your database grows beyond being your personal productivity tool and begins to be useful for others, or when your database operations become complex, you will find making an application has several benefits. For example:

- People unfamiliar with computers or data processing need little or no training to begin using an application for database operations.
- People with data processing knowledge can use applications to accomplish routine tasks without having to remember details.
- People using the database in a complex way, during which they must make keyboard entries in some particular order, can be guided through the correct order by the application displays.

To bring an application into existence, you or someone else has to design and build the system of menus, screens, and commands that connect the operator with the database. R:base offers you the EXPRESS, a tool that helps you quickly build productive applications, even if you have never built an application before.

### **Application Operator, Owner, and Developer**

In designing an application, keep in mind that there are three distinct roles: operator, owner, and developer. Previously, the developer and the owner in most cases were different people, since application building was a time-consuming and intricate job. Now the EXPRESS makes application building quick and easy so that you can be both owner and developer. In most cases, you will also be one of the operators.

An application operator uses a computer to run an application by typing at the keyboard and watching the screen for responses. See figure 2-12. The operator does not need to know about the configuration of the application system. To the operator, an application is the display screens that it generates. The operator is primarily concerned that the application provide information on what tasks are available to do next and when a task is complete.

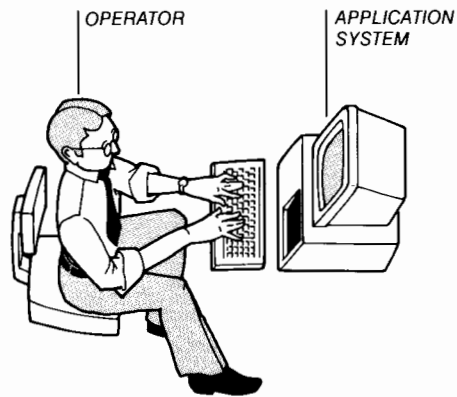


Figure 2-12 An Operator's View of an Application System

The database owner is responsible for the integrity and security of the data contained in the database and the database structure. See figure 2-13. The owner is primarily concerned with making sure that the application does not compromise the integrity of the database—for example, that the application prevent an operator from adding invalid data to the database or from deleting a table. The database owner is also concerned that the application prevent unauthorized operators from retrieving confidential data.

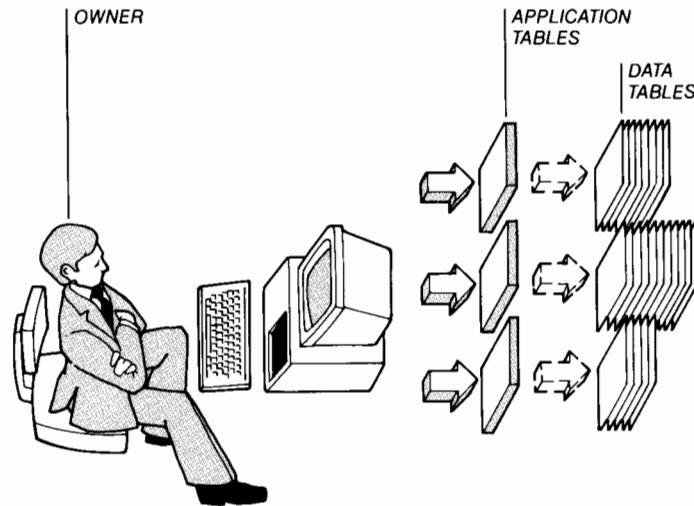


Figure 2-13 A Database Owner's View of an Application System

An application developer plans how the application operator, the application, the database, and the application owner will work together as a system. See figure 2-14. After planning, the application developer assembles available application components—menus, screens, command files, and the tables that hold forms and reports—and, if a needed component is not available, tailors one to the task.

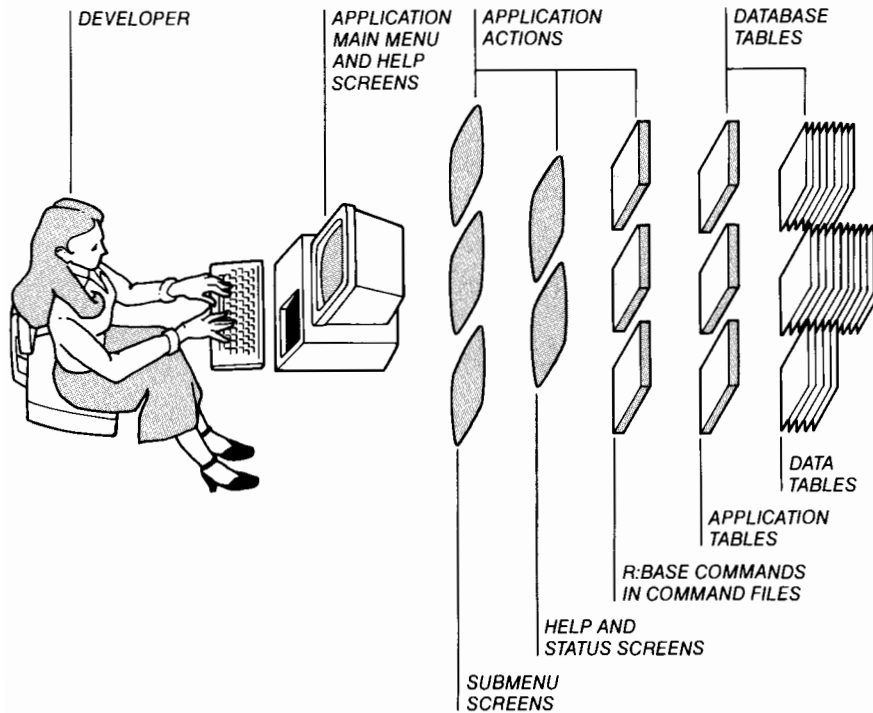


Figure 2-14 A Developer's View of an Application System

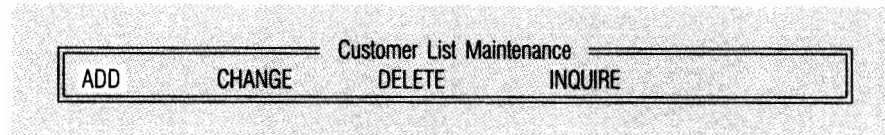
## Application Design with the Application EXPRESS

When you begin to design your application, make an initial sketch of the parts and their organization on paper before you use the EXPRESS to build them on the computer.

To see how you can use the EXPRESS for application building, consider again the example of the COMPUCO Sales department. Sales manager June Wilson, the COMPUCO database owner, decides she wants an application that will make it possible for others in her department to use her database. The application will provide menus so that an operator can maintain the data in three tables of the database: *salesrep*, *custlist*, and *product*. The basic maintenance functions the sales manager wants included are adding new rows to a table, changing data already in a table, deleting rows from a table, and displaying information on the screen. The application also should help the operator print customer lists, mailing labels, sales-representative lists, and product lists. The sales manager also wants the application to serve her sales representatives in building price quotes for customers either during a telephone call or immediately afterward.

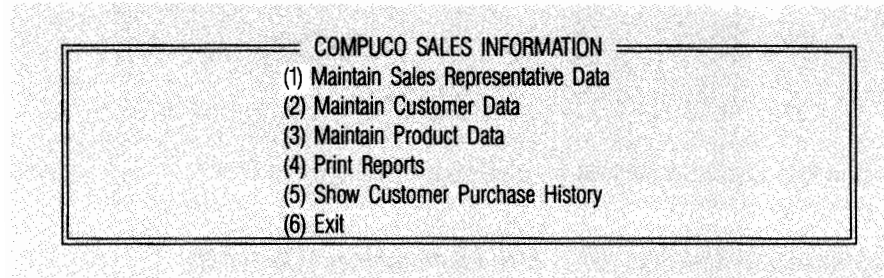
## Designing Menus

Decide how many menus you need for your application and what text you want in them. Then decide if you want horizontal or vertical menus. Horizontal menus place text across the screen in rows. An operator chooses a menu option by highlighting the option text itself with the cursor and pressing the [ENTER] key. A horizontal menu looks like this:



A horizontal menu may have multiple rows to accommodate more menu choices than are shown in this example. There may be as many as 40 choices in a multi-row horizontal menu.

Vertical menus allow you to enter more information for each choice. For example, the sales manager may want a vertical menu that looks like this:



A vertical menu can hold up to nine choices. The text, however, that describes each choice can be more detailed than for a horizontal menu. A vertical menu is therefore a good choice for use with untrained operators and for the main menu.

As in the case of designing a database, it is useful to map out the structure of an application system by sketching a menu tree. In general, you start with the main menu, which is the first screen the operator will see. Designing this menu is the first step in designing an application. All other parts of the application fan out from this menu—something like branches on an upside-down tree. Figure 2-15 shows a system of menus and submenus.

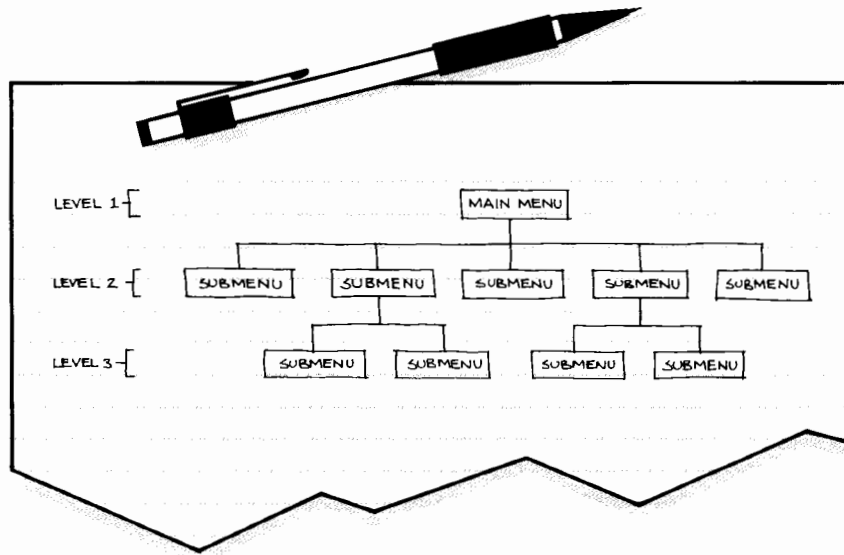


Figure 2-15 A Menu Tree

Follow these steps to make your sketch:

- Sketch what the application main menu will look like. Include all the important functions the application will perform, whether you know how to implement that function or not. Jot down the text for each menu choice.
- Look at every choice on the main menu and decide if the first action for that choice will be to display another menu (a submenu). If so, sketch it and jot down the text for the submenu choices. Draw a line from the main menu choice to the submenu. You have started to draw a menu tree. Note, however, that not all applications require submenus.
- Once you have sketched a menu tree for your application, name each of the individual menus. For submenus, use a naming convention so the name of the menu will imply which higher-level menu choice leads to its display.

Figure 2-16 shows a sketch of the menu tree for the example COMPUCO application. The main menu, *compmain*, has five choices. Choice (5) is included in the main menu even though it does not lead to a submenu display and even though the database owner does not plan to build a customized command file to implement this choice immediately.

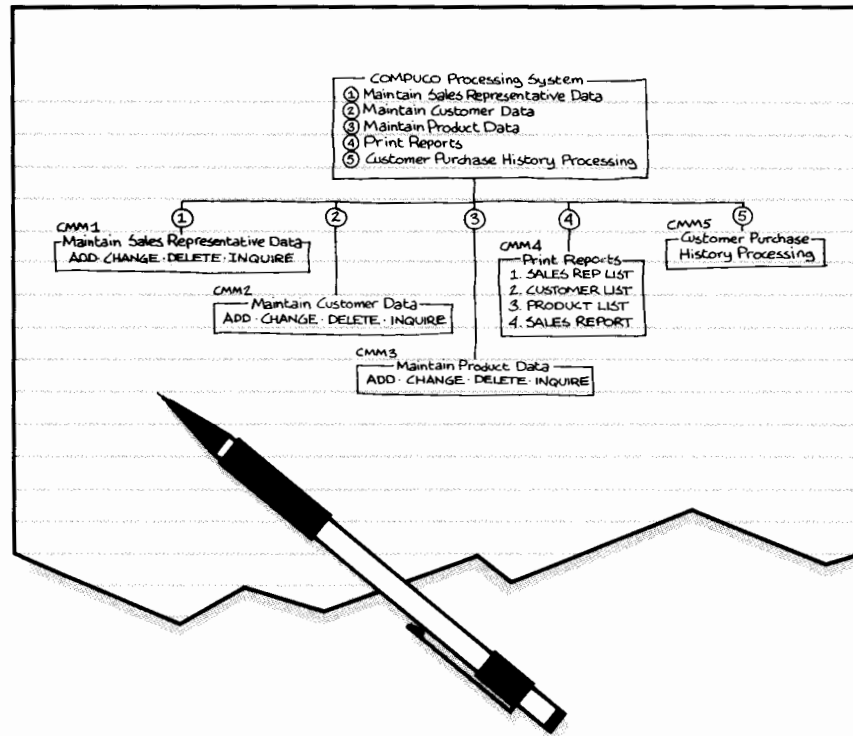


Figure 2-16 COMPUCO Application Menu Tree

The submenu names *CMM1*, *CMM2*, *CMM3*, *CMM4*, and *CMM5* indicate the number of the main menu choice which causes them to be displayed. The submenu title text also indicates the main menu choice that causes the submenu to be displayed.

With a menu tree sketch in hand similar to the one shown in figure 2-16, you can begin using the EXPRESS to build your application. The information you need to respond to the EXPRESS prompts in order to build your application main menu is included in the menu tree sketch.

The EXPRESS helps you with each choice, guiding you through the process of defining your screens and making them into a working application. The process is fast and easy.

## THE NEXT STEPS

With a good database design and good application design ready, use R:base 5000 to implement the design. The two tutorials in chapters 3 and 4 provide guidance for hands-on experience with R:base 5000. Use these chapters with the product to become familiar with basic operations and with the kinds of things you can do with it. Then, as you begin to put R:base 5000 to work for you, continue to explore the many features it has. Part 2 of this manual provides the information you need to use R:base fully for managing your database. When you are ready to make an application that will help you or others use your database, turn to Part 3. This part discusses application building in depth, showing you the full set of features the Application EXPRESS offers and the many tools R:base 5000 provides to help you make customized menus and macros for your applications.





---

## Introductory Tutorial Contents



<b>How to Use This Chapter</b>	3-3
<b>Getting Started</b>	3-3
Getting Help	3-4
Manuals	3-4
On-Line Assistance	3-4
The R:base 5000 Main Menu	3-5
<b>Defining a Database with the Application EXPRESS</b>	3-6
Leaving the EXPRESS	3-11
<b>Using R:base</b>	3-11
Entering R:base	3-12
Leaving R:base	3-12
<b>Opening a Database</b>	3-13
<b>Loading and Retrieving Data</b>	3-13
Entering Data	3-13
<b>Displaying Your Data</b>	3-14
<b>Looking at the Database Structure</b>	3-17
<b>Creating Tables from Existing Tables</b>	3-19
The PROJECT Command	3-19
The INTERSECT Command	3-20
<b>Making a Data Entry Form</b>	3-21
Designing an Entry Form	3-21
Laying Out Your Form	3-22
Locating Areas for Data Entry	3-24
<b>Entering Data with a Form</b>	3-26
<b>Preparing a Report</b>	3-28
Naming the Report	3-29
Defining Variables	3-31
Locating Columns and Variables	3-32
Marking Heading, Footing, and Detail Lines	3-33
Other Report Writer Options	3-35
Leaving the Report Writer	3-35

<b>Using Your Report</b>	3-37
Screen Display	3-37
Printed Report	3-37
File Report	3-38
<b>Maintaining Your Database</b>	3-38
Editing Data	3-39
Adding a Column to a Table	3-40
Deleting Rows	3-41
Removing Tables	3-41
<b>Leaving R:base</b>	3-41
<b>The Next Steps</b>	3-42

## HOW TO USE THIS CHAPTER

This chapter shows you how to use R:base 5000 for carrying out fundamental database operations, such as defining a database, loading it with data, retrieving data from the database, and making changes to the database and its data.

Work your way through all of this brief tutorial. When you finish, you will have hands-on experience with R:base 5000, and you will know enough to begin putting it to work.

Before you continue with this tutorial, it is recommended that you read chapter 2, which explains what a database is and also describes in detail the COMPUCO example which you will be using here.

Before you begin, prepare in this way:

- If your system has a hard disk and at least 320K: follow the installation, configuration, and start-up instructions in chapter 1. Then follow all directions in the tutorial.
- If your system has a hard disk and 256K to 320K RAM: follow the installation, configuration, and start-up instructions in chapter 1. Start the EXPRESS and R:base programs as described in chapter 1.
- If your system has two floppy drives: follow the start-up and configuration instructions in chapter 1. In this tutorial, pass over the section "The R:base 5000 Main Menu" and other instructions for use of the R:base 5000 main menu. Start the EXPRESS and R:base programs as described in chapter 1.

## GETTING STARTED

The quickest way to learn how to use R:base 5000 is to start using it. This tutorial gets you started. It shows you how to create a database and how to put data into it. Then, working with the COMPUCO database, you are shown the basic skills you need to make use of several key features of R:base 5000.

Take your time, use this tutorial for practice until you feel comfortable with R:base 5000. Remember that making errors is part of learning, and, among other things you will see while using this tutorial, it is easy to correct any errors you make with R:base 5000.

## Getting Help

Every part of R:base 5000 that you use in this tutorial is explained in detail in the manuals and in the on-screen help. Turn to these when you need additional help. Since this tutorial presents only an introductory glimpse of the many features of R:base 5000, you will find in the manuals and the help screens information about many commands and functions that are not covered in this tutorial.

## MANUALS

R:base 5000 comes with two books, the *R:base 5000 User's Manual*, which includes this tutorial, and the *R:base 5000 Reference Manual*. The second part of the *R:base 5000 User's Manual* provides detailed explanations of database operations that you will be introduced to in this tutorial. Keep the *R:base 5000 Reference Manual* close at hand. It contains a summary description of every R:base 5000 command. As you become familiar with the product, you will find this book a handy guide for quick reference.

## ON-LINE ASSISTANCE

When you are using R:base 5000, you can get on-line assistance. R:base 5000 contains a complete set of screen-explanations to provide the help you need. To display a help screen, press [F10].

To bring up the R:base 5000 main menu, at the C> prompt, type *RB5000*. Then press [ENTER].

```
*****  
*****  
*****  
*****  
*****  
*****  
*****  
*****  
*****  
*****  
*****  
*****  
*****  
*****  
*****  
*****  
  
*****  
*****  
*****  
*****  
*****  
*****  
*****  
*****  
*****  
*****  
*****  
*****  
*****  
*****  
*****  
*****  
  
*****  
*****  
*****  
*****  
*****  
*****  
*****  
*****  
*****  
*****  
*****  
*****  
*****  
*****  
*****  
*****  
  
*****  
*****  
*****  
*****  
*****  
*****  
*****  
*****  
*****  
*****  
*****  
*****  
*****  
*****  
*****  
*****
```

R:BASE SERIES 5000

Copyright (c) 1985  
by MICRORIM, Inc.  
Bellevue, WA

```
===== Select an option, or press [ESC] to return to DOS =====
EXPRESS  GATEWAY  RBASE    RBEDIT    RCOMPILE  CLOUT2
```

This is the main menu. The options listed are:

- EXPRESS: the Application EXPRESS
- GATEWAY: the FileGateway, R:base 5000's file-transfer program
- RBASE: the R:base database management system
- RBEDIT: the R:base file editor
- RCOMPILE: the compiler used for application files
- CLOUT: Microrim's natural language inquiry option

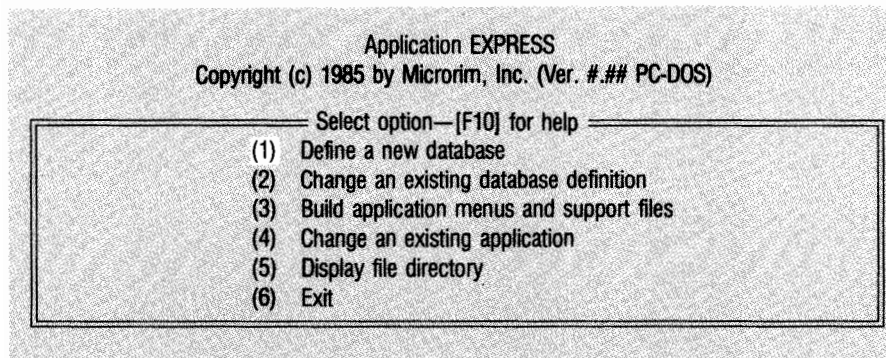
In this tutorial, you will be using two parts of R:base 5000: the EXPRESS and R:base. The EXPRESS contains a defining option that will help you quickly set up the structure of your database. R:base contains the commands that you use to enter data, to maintain it, and to retrieve data for screen displays and in printed reports.

## DEFINING A DATABASE WITH THE APPLICATION EXPRESS

R:base stores your information in the form of a table. Before you begin entering the data you want stored, you first define the database by giving it a name. You then name a table that you want in the database and name the columns you want in the table. You then name the next table and its columns and continue until you have named all tables and columns you want in the database. The EXPRESS makes it easy to define your database.

This section of the tutorial shows you how to use the EXPRESS define screens. It provides directions for you to define a database named *mydata* and to define two tables and their columns. The table and column names are similar to those used in the example in chapter 2.

To begin, choose the EXPRESS from the main menu. Use the left-arrow or right-arrow key to place the lighted block on the word EXPRESS. Then press [ENTER]. The EXPRESS main menu comes on screen.



To make a selection, you can press a key for the number. You can also use the down-arrow key, the up-arrow key, or the space bar to highlight the number in the menu and then press [ENTER].

For now, press [1] so you can begin defining your database.

The first screen asks you to name your database.

Enter your database name (1-7 characters)

The name can contain no more than seven characters. Name it *mydata*.

When you press [ENTER] the EXPRESS screen for defining tables is displayed. As you define the table name and column names, the table headings fill up to display the names and data types you have chosen.

Enter the name for this table

← TABLE NAME GOES HERE

↗ COLUMN NAMES GO HERE


The cursor, in the upper left box, shows where to enter your table name. Table names can contain no more than eight characters. Name the first table *custlist* (short for customer list). It will store data for a simple phone list, and it will have three columns: *custid* (for customer identification numbers), *company* (for company names), and *phone* (for company telephone numbers).



### 3-8 Getting Started with R:base 5000

---

With the cursor in the table-name box, type *custlist*.

Before you press [ENTER], check the entry. If it is correct, press [ENTER]. If not, you can correct it by moving the cursor left or right to the character you want to change. Then type over it. You can also use the backspace key.

The cursor moves to the head of the first column, ready for your first column name. Like table names, column names can contain no more than 8 characters.

Type *custid* and press [ENTER].

The EXPRESS displays the list of R:base data types beneath the columns of the table.

Select column data type—[ENTER] to choose					
TEXT	DOLLAR	INTEGER	REAL	DATE	TIME

Since the customer identification numbers in this sample table will be whole numbers only, choose INTEGER. You can press [I] or move the lighted bar to INTEGER with the right-arrow key, the tab key, or the space bar. Then press [ENTER].

The cursor moves to the next column, ready for the entry of the column name. Type *company* and press [ENTER].

*Company* is TEXT, so with the selection bar on TEXT, press [ENTER].

Because it is a text field, the EXPRESS prompts you to define the maximum column width. It supplies a default number of 8. You need a longer length, however. In this sample database, company names are not expected to exceed 40 characters (including spaces).

Type *40* and press [ENTER].

Now, define the last of the three columns in the table. For the column name, type *phone* and press [ENTER]. For the data type, choose TEXT with the length of 12. Press [ENTER] to make the selection. Using TEXT for the data type allows you to include spaces or dashes in the entry of a telephone number—for example, 206-641-6619. If you had chosen INTEGER, you would have to enter numbers only, like this: 2066416619.

Your completed sample table looks like this:

custlist						
custid	company	phone				
INTEGER	TEXT 40	TEXT 12				

Database mydata --- Defining table custlist --- Column 4

To store the completed table, press [ESC].

The next screen that appears gives you the options of continuing to define tables, changing the definition of the table you defined, removing it from the database, or exiting to the EXPRESS main menu.

Press [1] so you can add another table to your database.

When the define screen returns, name the next table *salesrep* (short for sales representative). Then give the table these seven column names and types:

empid	INTEGER	
frstname	TEXT	10
lastname	TEXT	16
address	TEXT	30
city	TEXT	20
st	TEXT	2
zip	TEXT	10

Notice that when you made the entries for *zip* that the display of columns shifted to the left, leaving a blank column at the right of your screen, like this:

salesrep						
frstname	lastname	address	city	st	zip	
TEXT 10	TEXT 16	TEXT 30	TEXT 20	TEXT 2	TEXT 10	

Database mydata --- Defining table salesrep --- Column 8

If you continue to add columns, the column farthest left leaves the display to make room for another entry on the right. You always know where you are, however, by observing at the bottom of the screen the information message that indicates the next column to be defined in your table.

The next column, column 8, is to be *phone*. Since you stated its data type and length when you placed it in the *custlist* table, the EXPRESS enters that part of the information for you. Type *phone* and press [ENTER] to see how it works.

Name column 9 *hiredate*. For its type, choose DATE.

Since *hiredate* is the last column you need, press [ESC] to store the table.

You now have a database named *mydata* with two tables named *custlist* and *salesrep*. Your database has a total of eleven different column names—*custid*, *company*, *phone*, *empid*, *frstname*, *lastname*, *address*, *city*, *st*, *zip*, and *hiredate*. The database, however, has a total of twelve columns, since *phone* is present in both tables.

Press [4] to leave the EXPRESS's table-define screens. The EXPRESS returns you to its main menu.

Choice 5 on the main menu is *Display File Directory*. To see a list of files, press [5].

The EXPRESS asks you to enter the DOS filenames you want to see. Press [ENTER] to display all files.

Among other files listed from your current directory, you will see these three files:

C:MYDATA1.RBS C:MYDATA2.RBS C:MYDATA3.RBS

(The initial letter C: is the drive designator. If you are using a two floppy drive system, it will be B:.)

The three files hold the database you created. At this point, it is not necessary for you to learn the significance of these three files. You should remember, however, that when you want to work with this database, R:base 5000 needs to be able to reach all three.

When you are finished looking at the directory, press any key to return to the main menu.

## Leaving the Express

From the EXPRESS main menu, press [6] to exit.

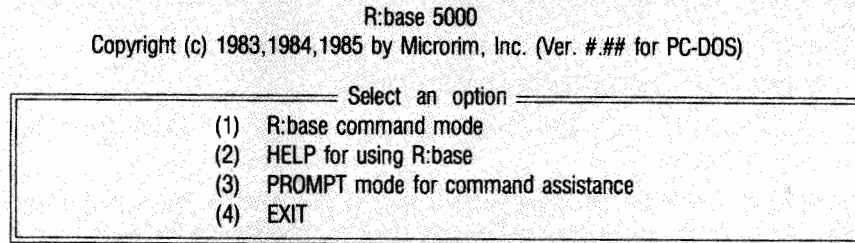
You are now back to the R:base 5000 main menu.

## USING R:BASE

Once you have defined your database and the tables you need, you are ready to enter data into the database and to use R:base commands for viewing your data and retrieving information for displays and reports. This section of the tutorial will show you how to do these basic operations with R:base.

## Entering R:base

From the R:base main menu, choose *RBASE*. When you press the [ENTER] key, the first R:base menu appears.



Your options are:

- R:base command mode: Enter R:base commands
- HELP: Display help screens for R:base commands and functions
- PROMPT mode: Provide on-screen instructions and assistance for carrying out commands
- EXIT: Leave R:base and return to R:base 5000 main menu

Choose (1) so you can try out R:base commands.

Notice the R> prompt that appears. When this prompt is on the screen, you can enter an R:base command. To execute a command you have typed, you press [ENTER]. The next section shows you how to use some basic commands when you want to work with your database.

## Leaving R:base

When you are through working with R:base, you can leave R:base any time the R> prompt is displayed. You can exit in two ways:

- Press [ESC]. Pressing [ESC] returns you to the R:base main menu.
- Type *EXIT* and press [ENTER]. The EXIT command returns you to the R:base 5000 main menu.

Be sure to exit in one of these two ways. If you do not exit properly, you may lose valuable data.

## OPENING A DATABASE

To work with your database, you tell R:base to open it. You do this by using the OPEN command. At the R> prompt enter the command like this:

```
OPEN mydata
```

Include the drive or path if the database is not on your current directory—for example, *c:\mydir\mydata*.

When R:base finds your database, it responds with this message: *Database exists*. You can now use R:base to work on your database.

Note that throughout the manual, the commands are shown with the part that you use for all entries in upper-case letters and the part you change, such as a database name, in lower-case letters. You can, however, use upper-case or lower-case letters or a mixture of both in your commands.

## LOADING AND RETRIEVING DATA

### Entering Data

R:base offers you several methods for entering data into your database. A good method while you are learning is to use the LOAD command with prompts. This command guides you through data entry. The screen displays a column name and data type to help you load data successfully.

Try using the LOAD command with prompts to enter a few rows of data into the *custlist* table. At the R> prompt, type:

```
LOAD custlist WITH PROMPTS
```

After you press [ENTER], R:base displays the entry screen. Follow the instructions and enter data so that your screen looks like this:

```
R>LOAD custlist WITH PROMPTS
Begin R:base Data Loading

Press [ESC] to end, [ENTER] to continue
custid      (INTEGER) :100
company     (TEXT    ):PC Distribution Inc.
phone       (TEXT    ):617-341-2189

Press [ESC] to end, [ENTER] to continue
```

Now, add two more rows. Enter this data and again follow the prompts R:base provides.

```
custid: 101
company: Computer Distributors Inc.
phone: 617-423-8921
```

```
custid: 102
company: Industrial Computers Inc.
phone: 303-239-7823
```

If, after you press [ENTER], you discover you made an entry error, you cannot fix it while you are loading data. For now, do not worry about errors. Later, in this tutorial, you will see how easy it is to make corrections with the EDIT command.

You now have three rows into your *custlist* table. Press [ESC] to return to the R> prompt.

## Displaying Your Data

Use the SELECT command to see the data you have entered. You can enter this command whenever the R> prompt is displayed. Try the following basic uses of the command.

To display all columns and rows from the *custlist* table, at the R> prompt enter the command like this:

```
SELECT ALL FROM custlist
```

R:base displays the data from the table on your screen like this:

```
R>SELECT ALL FROM custlist
custid  company                                phone
-----
    100  PC Distribution Inc.              617-341-2189
    101  Computer Distributors Inc.        617-423-8921
    102  Industrial Computers Inc.          303-239-7823
```

To display particular columns, enter the column names in the command. For example, entering the command like this will display the data for the *company* and *phone* columns only:

```
SELECT company phone FROM custlist
```

You can have R:base sort the data by adding a **SORTED BY** clause to the command, like this:

```
SELECT company phone FROM custlist SORTED BY company
```

This clause tells R:base to list the company names in alphabetical order. When you enter the command, the display looks similar to this:

```
R> SELECT company phone FROM custlist SORTED BY company
```

company	phone
Computer Distributors Inc.	617-423-8921
Industrial Computers Inc.	303-239-7823
PC Distribution Inc.	617-341-2189

You can set conditions for the selection of data with a **WHERE** clause. You place in the clause the criteria that you want satisfied for a row to be displayed on the screen. You can have up to ten conditions. For now, however, try this simple example. Tell R:base to select only the companies with telephone area code 617. You do it by adding this clause to the command: *WHERE phone EQ 617\**. *Phone*, of course, is the column name where R:base is to look for the numbers. *EQ* tells R:base that the phone number must have the numbers (and characters) you specify. The asterisk (\*) after 617 tells R:base to look for all phone numbers that begin with 617 and that have any subsequent characters in the string, such as -423-8921. You can use both a **SORTED BY** clause and a **WHERE** clause in the same command, so, at the R> prompt, enter the command like this:

```
SELECT company phone FROM custlist SORTED BY company WHERE phone EQ 617*
```



R:base displays a sorted list with the columns you named and rows that meet the condition you specified. The display on your screen looks similar to this:

```
R>SELECT company phone FROM custlist SORTED BY company WHERE phone EQ 617*
company                                     phone
-----
Computer Distributors Inc.                 617-423-8921
PC Distribution Inc.                       617-341-2189
```

You can use a table's column names in your **WHERE** clause even if they are not part of the display. For example, look at this **SELECT** command:

```
SELECT company phone FROM custlist SORTED BY company +
WHERE phone EQ 617* AND custid LT 101
```

This entry of the command has several features to note.

- It has become so long it has to be spread over two lines. When your command line needs more than 78 characters, put a space after any character up to the seventy-seventh character, type a plus sign (+), and then press [ENTER]. R:base displays another R> prompt on the next line, and you can continue entering the command. See the screen example below.
- Its **WHERE** clause contains two conditions connected by *AND*. This means that, to be displayed, the row must satisfy both conditions.
- The second condition—that the company identification number be less than 101—is to be met, although the customer numbers are not included in the display.

Enter the command to see the results. Your screen display looks similar to this, since the row for only one company satisfies both conditions:

```
R>SELECT company phone FROM custlist SORTED BY company +
R>WHERE phone EQ 617* AND custid LT 101
company                                     phone
-----
PC Distribution Inc.                       617-341-2189
```

## LOOKING AT THE DATABASE STRUCTURE

To use the rest of the commands described in this tutorial, you will find it helpful to have a larger database with which to practice. Your R:base 5000 product disks include a sample database called *compuco*. This database has been built according to the design explained in chapter 2. It contains seven tables. Each table has several rows of data.

You can work with only one database at a time. When you want to change from one database to another without leaving R:base, use the OPEN command to open the database you want to work with next. R:base automatically closes *mydata*, the database you had open, and stores your data.

At the R> prompt, type:

```
OPEN compuco
```

With the *compuco* database now open, take a look at its structure—the tables and columns it has. The command to use is LIST, which displays information about the database. The variations of the command allows you to see information about all columns, all tables, or about a particular table. Try the following forms of the command.

### LIST TABLES

R:base displays the names of the seven data tables in the COMPUCO database, including REPORTS, a special table that holds reports added to the database. Your screen looks similar to this:

```
R>LIST TABLES
Tables in the Database COMPUCO
```

Name	Columns	Rows	Name	Columns	Rows
transx	7	41	compnent	2	12
product	7	27	salesrep	10	5
custlist	9	9	compused	2	67
REPORTS	2	68			

```
R>
```

LIST salesrep

R:base will display the names, the data types, and the lengths of the columns in the table *salesrep*. The information includes some features that are not discussed in detail in this tutorial. In brief, they are:

- Passwords: you can use password-protection for your tables, so that a user has to enter a password to read data from a table or to make changes to the data.
- Keys: you can make a column keyed to increase the speed with which R:base will process the data from the column. R:base allows you to make the column keyed when you define it or when you use the BUILD KEY command. You may key as many columns as necessary.

R>LIST salesrep

Table: salesrep

Read Password: NO

Modify Password: NO

Column definitions

#	Name	Type	Length	Key
1	empid	INTEGER	1 value(s)	
2	firstname	TEXT	10 characters	
3	lastname	TEXT	16 characters	
4	address	TEXT	30 characters	
5	city	TEXT	20 characters	
6	state	TEXT	2 characters	
7	zip	TEXT	10 characters	
8	phone	TEXT	12 characters	
9	ext	TEXT	3 characters	
10	hiredate	DATE	1 value(s)	

Current number of rows: 5

## CREATING TABLES FROM EXISTING TABLES

R:base has six commands that allow you to combine tables: PROJECT, APPEND, UNION, INTERSECT, SUBTRACT, and JOIN. This section shows you two of them: PROJECT and INTERSECT.

### The PROJECT Command

The PROJECT command makes a new table that contains some—or all—of the columns and rows from a table you specify. To try out the command, make a table from *custlist* that has two columns: *company* and *phone*. In the *compuco* database, the *custlist* table contains columns for a customer identification number, a contact's first and last names, a company name, address, and telephone number. The new table will simply be a customer telephone list, so it can be named *cstphone*. Adding the SORTED BY clause will arrange the new table in alphabetical order by company.

At the R> prompt, enter the command like this:

```
PROJECT cstphone FROM custlist USING company phone SORTED BY company
```

R:base carries out the command and gives you a message when the command is completed. Since the command was not limited with a WHERE clause, all rows in the *custlist* table are transported to *cstphone*, the new table. To display the data in the new table, use the SELECT command. Your screen will look similar to this:

```
R> PROJECT cstphone FROM custlist USING company phone SORTED BY company
Successful project operation      9 rows generated
R> SELECT all FROM cstphone

company                           phone
-----
Computer Distributors Inc.        617-423-8921
Computer Mountain Inc.            303-271-1500
Computer Warehouse                 812-701-1002
Industrial Computers Inc.          303-239-7823
Industrial Concepts Inc.           415-878-5600
Midtown Computer Co.              312-277-5000
PC Consultation and Design         415-892-6745
PC Distribution Inc.               617-341-2189
Southwest Computers, Inc.         512-581-8800
R>
```

## The INTERSECT Command

The INTERSECT command makes a new table from two tables that already exist in your database. For the command to work, your tables have to have at least one column name in common. The new table formed has all the column names from the other two tables but without repeating common column names. The rows are selected only if the same data appears in common columns. (The data in a column is referred to as a *value* throughout this manual).

In the *compuco* database, the *transx* table has identification numbers for the company, the computer, and the sales representative. Suppose you want to see names rather than numbers. The solution is to use the INTERSECT command to make a new table that gives you this information. For example, to make a table that will have the computer names, combine the *transx* table with the *product* table. It is to be stored only temporarily, since you will not want to update the data in this table and in the other two tables, so call it *temp1*. At the R> prompt, enter the command like this:

```
INTERSECT transx WITH product FORMING temp1
```

The new table contains all the columns from *transx* and *product*. To see the results, select a few columns and rows from *temp1*. For example:

```
R> INTERSECT transx WITH product FORMING temp1
Successful intersect operation      41 rows generated
R> SELECT prodname price cost FROM temp1 WHERE custid = 101
```

prodname	price	cost
Superior Color PC	\$2,875.00	\$2,130.00
Standard PC	\$1,800.00	\$1,400.00
Standard PC with deluxe keyboard	\$1,600.00	\$1,430.00
Advanced PC with deluxe keyboard	\$2,300.00	\$1,830.00
Superior PC with deluxe keyboard	\$2,575.00	\$2,030.00
Portable Standard PC w/deluxe kbd	\$3,100.00	\$2,225.00
Portable Classic PC	\$2,700.00	\$2,300.00

```
R>
```

You can easily remove tables you no longer need. See "Maintaining Your Database" later in this tutorial.

## MAKING A DATA ENTRY FORM

Rather than using the LOAD command, you may find it handy to enter data with an entry form. In fact, you can have more than one form for every table. For now, make a form that will enter data into the *salesrep* table.

### Designing an Entry Form

When you are learning to make forms with R:base, it is a good idea to do some preparation. First, look at the structure of the table with the LIST command. At the R> prompt, type:

```
LIST salesrep
```

As you can see on the screen display, the table has ten columns. You need to allow a place on the form where the data for each can be entered.

When you are learning, it is also helpful to have some idea of what the form is to look like. R:base lets you use the whole screen for the form, so your basic limitation in size is only the screen. For this form, use the design shown in figure 3-1. It has two lines at the top to identify its purpose, and for each column entry, it has complete words to make clear what the entry is for.

SALES REPRESENTATIVE INFORMATION FORM			
Employee Identification No.:			
First name:	Last name:		
Home address:			
City:	State:	Zip:	
Home telephone:			
Work telephone extension:			
Date hired:			




Figure 3-1 A Design for a Table Entry Form

## Laying Out Your Form

Use R:base FORMS mode to make the form. You enter the FORMS mode by typing *FORMS* at the R> prompt and leave it by selecting *QUIT* from the FORMS menu. Within FORMS, you usually first design the form by typing in the titles and labels you want to use, and then you tell R:base where you want the entry block for each column you are using.

At the R> prompt, type:

```
FORMS
```

R:base asks you for a name for the form. The name can have up to eight characters. Type in this name:

```
repform
```

R:base, in effect, then offers you the choice of a table form or a variables form. In this example, you are developing a table form, so enter the table name so that the screen looks like this:

```
R> FORMS
Begin R:base forms definition
Enter form name:repform
Enter table name (for variables form press [ENTER]):salesrep
```

R:base displays the FORMS menu.

```
-----Edit-----Locate-----Quit-----
```

To draw your form on the screen, choose **EDIT**. You can type **[E]** or, if *Edit* is highlighted, press **[ENTER]**. R:base then displays a screen on which you can type in the text you want to appear on your form.

Look at the top right of the screen.

< 1, 1> [F3] to list, [ESC] to exit

- < 1, 1> is the row and column counter. It will tell you where your cursor is located.
- Pressing **[F3]** will display a list of the columns in the *salesrep* table.
- Pressing **[ESC]** will take you back to the previous screen.

This informational line will not be displayed on your form when you use the form for data entry.

Lay out the form by typing in the text from the design. The following screen shows which lines and columns to use so that you leave adequate space for data entry.

You can move the cursor around the screen with the cursor-arrow keys. You can also use the keys listed in table 3-1 to speed your preparation of the form and to make changes or corrections.

Table 3-1 Editing Keys for Forms

<b>[Del]</b>	Removes the character at the current cursor position.
<b>[Ins]</b>	Inserts a space at the current cursor position.
<b>[F1]</b>	Inserts a line at the current cursor position and moves all following lines down one line.
<b>[F2]</b>	Deletes a line at the current cursor position and moves all following lines up one line.
<b>[F3]</b>	Lists the columns in the table.
<b>[F4]</b>	Toggles the repeat mode ON and OFF. In the repeat mode, the last character entered is repeated when the cursor is moved.



START AT <1,30> → SALES REPRESENTATIVE  
START AT <2,32> → INFORMATION FORM

Employee Identification No.:

First name: <6,25> → Last name:

Home address:

City: <8,29> → State: Zip:

Home telephone:

Work telephone extension:

Date hired:

When you finish, press [ESC].

## Locating Areas For Data Entry

Next, you need to locate where data for each column are to be entered. To do this, you tell R:base which column you are locating, you place the cursor where you want the entry to begin and type [S] (start). At the end of the entry space for the column, type [E] (end).

From the main menu, choose *Locate*.

R:base asks you for the name of the column you want to locate on the form. Locate the columns in order. Table 3-2 lists the column names and the corresponding labels on the form.

Table 3-2 *Salesrep* Columns and Data-Entry Form Labels

Column Name	Data-Entry Form Label
empid	Employee Identification No.
frstname	First name
lastname	Last name
address	Home Address
city	City
state	State
zip	Zip
phone	Home telephone
ext	Work telephone extension
hiredate	Date hired

Remember that you can also see a list of columns by pressing [F3].

Type *empid* and press [ENTER].

Follow the screen instructions to move the cursor to where you want the data entry for that column to start. Place the cursor two spaces after *Employee Identification No.:* and press [S]. R:base automatically moves the cursor to the end of length for *empid*. Press [E].

The line looks like this on the screen:

```
Employee Identification No.: s          e
```

Continue to locate the data entry spaces for columns until your completed form looks similar to this on the screen:

```

                                SALES REPRESENTATIVE
                                INFORMATION FORM

Employee Identification No.: s      e

First name: s      e Last name: s      e
Home address: s      e      e
City: s      e State: se Zip: s      e

Home telephone: s      e
Work telephone extension: s e

Date hired: s      e

```

You are now ready to leave the FORMS mode, so press [ESC] and then [Q] to quit. R:base displays this menu:

```

---Save changes---Discard changes---Return-----

```

Press [S] for *Save changes* to store your form.

## ENTERING DATA WITH A FORM

Now that you have made a form, you can use it whenever you want to enter data into the *salesrep* table. Try it out.

At the R> prompt, enter this command:

```
ENTER repform
```

Your form appears on the screen, and you are prompted to enter data for columns in the order in which you located the columns on the form. The lighted block indicates where you can type in data. After you type in data for a column, press [ENTER]. Work your way through the form, so that when you finish, it looks like this:

```

Press [ESC] when done with this data
                                SALES REPRESENTATIVE
                                INFORMATION FORM

Employee Identification No.: 200

First name: Thomas   Last name: Green
Home address: 100 Maple Ave.
City: Bellevue       State: WA Zip: 98004

Home telephone: 206-234-5678

Work telephone extension: 101

Date hired: 2-4-85

```

Press [ESC]. R:base now presents a list of options.

```

--Add--Reuse--Edit--Quit-----

```

You want to add this data to the table, so press [A].

Now you can add more data or quit. Here is how you quit. First, press [ESC]. Then press [Q] to quit.

When the R> is displayed, use the SELECT command to see the table with your addition. The table is too wide for all columns to fit on the screen at once, so choose a few, like this :

```
SELECT firstname lastname address city FROM salesrep
```

## PREPARING A REPORT

With the R:base report writer you can create a variety of reports, such as form letters, sales and expense reports, and statements. You can print information on preprinted forms, such as checks, invoice forms, and insurance forms.

Once you have prepared a report, you can display it on the screen, send it to your printer, or send it to a disk file.

This section introduces you to some of the many features of the report writer by having you prepare a simple report with the *compuco* database. As with preparing a form, you may find it helpful while you are learning to use the report writer to begin with a design for a report. Figure 3-2 shows how the sample report will look when you finish. Its purpose is to provide a weekly summary of transactions.

COMPATIBLE COMPUTER COMPANY WEEKLY TRANSACTION REPORT 03/15/85				
Date	Company	Computer	Units	Purchase Total
03/11/85	Industrial Concepts Inc.	CX3020	12	\$30,900.00
03/11/85	Industrial Concepts Inc.	MX3020	12	\$27,600.00
03/11/85	Industrial Concepts Inc.	PB3050	12	\$38,100.00
03/12/85	Industrial Concepts Inc.	PB3040	25	\$56,250.00
03/13/85	Industrial Concepts Inc.	MB3000	25	\$45,000.00
03/13/85	Industrial Concepts Inc.	CX3000	10	\$18,000.00
03/13/85	Industrial Concepts Inc.	PB3040	10	\$22,500.00
Weekly Total:				\$238,350.00

Figure 3-2 The COMPUCO Weekly Transaction Report

Like a form, a report basically uses one table. For this report, the table is *transx*. Notice, however, that only the columns *ldate*, which provides the data shown under *Date*, *prodid*, which provides the data shown under *Computer*, and *units*, are taken directly from that table. The report columns labelled *Company*, *Purchase Total*, and *Weekly Total* are developed through special features of REPORTS. These are explained below as you prepare the report.

## Naming the Report

To begin, type *REPORTS* at the R> prompt. You are then asked to name the report and to tell R:base which table you want to use for the report. Name the report *wklyrpt*. Type in the information so that your screen looks like this:

```
R>REPORTS
Begin R:base reports definition
Enter report name:wklyrpt
Enter table name:transx
```

R:base then displays the report menu.

```
Edit report---Define---Locate---Mark---Set---Help---Quit-----
```

From this menu, choose *Edit report*.

The screen that is displayed is the one you use for laying out the report. Although it looks similar to the forms-editing screen, you are not restricted to the size of the screen. For example, if you have a wide-carriage printer, you can make a report as wide as 131 columns. You can also use more than the 23 lines available on the screen. In this sample report, however, you do not need more than the screen you see.

For laying out your report, you can move the cursor around with the cursor-arrow keys. You can also use the keys listed in table 3-3.

Table 3-3 Edit Keyboard Functions

Key	Definition
[Ctrl] [→]	Moves the cursor to column 131 on the current line.
[Ctrl] [←]	Moves the cursor to column 1 on the current line.
[Del]	Deletes a space or character at the current cursor position.
[End]	Moves the cursor to the bottom of the report.
[Home]	Moves the cursor to the top of the report.
[Ins]	Inserts a space at the current cursor position.
[PgUp]	Moves the cursor up one page.
[PgDn]	Moves the cursor down one page.
[F1]	Inserts a line at the current cursor position and moves all following lines down one line.
[F2]	Deletes a line at the current cursor position and moves all following lines up one line.
[F3]	Lists the columns in the table.

Type in the text for your report so that it looks like this on your screen (do not be concerned about the reverse-video bar that extends down the left side of the screen; its purpose will be explained soon):

```

                                     < 1, 1> [F3] to list, [ESC] to exit
                                     COMPATIBLE COMPUTER COMPANY
                                     WEEKLY TRANSACTION REPORT
<1,26> →
<2,27> →
<5,1>  <5,14>
Date    Company
-----
                                     <5,45> <5,56> <5,64>
                                     Computer Units Purchase Total
                                     -----
                                     <8,64> →
<9,49> → Weekly Total:

```

When you finish, press [ESC] to return to the REPORTS main menu.

## Defining Variables

Three columns on this report and two other parts of it are going to use data that is not stored in the *transx* table. You use data not in *transx* by defining report variables. You tell R:base what value you want to go into the variable and you show on the report form where the variable value is to go.

The variables you need for this report are:

- The date you print the report. This will go under the title, like this:

COMPATIBLE COMPUTER COMPANY  
WEEKLY TRANSACTION REPORT  
03/15/85

The value you use for date is obtained from the system variable *#DATE*.

- The company name. This value comes from the column *company* in the table *custlist*. This process is called *lookup*, since R:base looks up the company name in another table and brings it into this report. The variable tells R:base the column and table and the condition for selection of data.
- Purchase total. This value is the result of a calculation: *units x price* (two of the columns in the *transx* table).
- Weekly total. This value is the sum of purchase totals.

You begin variable definition by choosing *Define* from the Report Menu.

R:base displays this prompt:

Expression:

[F3] to list, [ESC] to exit

For each variable you need a name. Enter the variables listed below one at a time. Be sure to leave one space on either side of the equals (=) sign. Press [ENTER] after you type in each of them.

wkdate = #DATE

cmpny = company IN custlist WHERE custid = custid

purtotal = units x price

wktotal = SUM OF purtotal



Note that R:base displays each variable as you define it. When you have defined all four variables, your screen looks like this:

```

Expression:                                     [F3] to list, [ESC] to exit

1:DATE      :   wkdate   =  #DATE
2:TEXT      :   cmpny    =  company  IN custlist WHERE custid = custid
3:DOLLAR     :   purtotal =  units    x  price
4:DOLLAR     :   wktotal  =  SUM      OF purtotal

```

If you find you made an error after defining a variable, define it again. When you press [ENTER], R:base will ask if you want your new variable to replace the one you entered earlier.

To return to the main menu, press [ESC] .

## Locating Columns and Variables

From the menu choose *Locate*.

Then choose *Locate* again from the Locate command-prompt menu. R:base prompts you to enter the name of a column or variable. For the report, you need these columns: *tdate* (transaction dates), *prodid* (computer numbers), and *units*. And you need these variables: *wkdate*, *cmpny*, *purtotal*, and *wktotal*. When you enter the name you are then asked to show where its data starts and ends on the report. As with forms, you mark these by pressing [S] at the beginning and [E] at the end.

Locate the columns and variables so your finished report layout looks similar to the following screen. Notice that the length is shorter for some columns and variables than the table or data type width. R:base allows you to use only the column width you need. You may notice too that when you locate *purtotal* and *wktotal* that the screen shifts to the right. This is because R:base is making room for you to end the space for the data. To return left, press the left-arrow key several times.

- Place *wkdate* on the third line. Move the cursor under the *T* of the word *TRANSACTION*. Press [S]. The cursor moves to the end of the data length. Press [E].
- Place *tdate* on line seven under the dashed line for *Date*. Place the cursor on line seven at column one. Press [S]. The cursor will move under the last of the eleven dashes. Press [E].
- Place *cmpny* under the dashed line for *Company*. Move the cursor under the first dash and press [S]. Move the cursor under the last dash and press [E]. (The [W] option is for long text entries that fill more than one report column line.)
- Place *prodid* under the dashed line for *Computer*. Move the cursor under the first dash and press [S]. Then move it as far as it will go—six spaces. This is the full length of *prodid*. Press [E], not [W].
- Place *units* under the dashed line for *Units*. Move the cursor under the first dash and press [S]. Move it under the final dash for *Units* and press [E].
- Place *prttotal* under the dashed line for *Purchase Total*. Move the cursor under the first dash and press [S]. Move it under the final dash and press [E]. R:base asks if you want this dollar data printed in check format. Press [N] for no.
- Place *wktotal* so that it starts at column 64 of line nine—under the dashed line on line eight. Move the cursor to column 64 and press [S]. Move the cursor under the final dash and press [E]. R:base asks if you want this dollar data printed in check format. Press [N] for no.

< 1, 1 > [F3] to list, [ESC] to exit

**COMPATIBLE COMPUTER COMPANY**  
**WEEKLY TRANSACTION REPORT**

Date		Company		Computer		Units		Purchase Total	
S	E	S	E	S	E	S	E	S	E
Weekly Total:								S	E

When you finish, press [ESC]. You return to this menu:

```
---Locate---Relocate---Help---Quit-----
```

Notice that you can choose to relocate. When you want to change the location of a table column or variable on your report, you choose this option.

For now, choose *Quit*.

## Marking Heading, Footing, and Detail Lines

Marking is the process you use to lay out the page as you want it printed. You tell R:base which lines you want at the top and the bottom and which lines are for information. For this report you use three types of lines:

- Heading for the top of a page
- Footing for the bottom of a page
- Detail for the central lines, which hold the information from the table and variables

From the Report Menu, choose *Mark*. R:base displays this menu:

```
--- Report--- Page--- Detail--- Break--- List--- Help--- Quit-----
```

The two choices you use for this report are *Report* and *Detail*. *Page* and *Break* are for reports that are more complex than this one.

Choose *Report*. R:base then asks a series of questions about page ejects. For now, pass up the options by pressing [N] at each question.

R:base then displays your report format on the screen and asks you to mark the heading lines. Notice that the cursor is in the left column. As you select heading lines by pressing [H], R:base inserts *HR* for report heading, and the cursor moves down the column.



The one line you have not marked is for detail—the data from *transx* and the variable *purtotal*. So choose *Detail* from the menu, and then type [D] to mark that line as shown below.

```

Press [D] to mark detail:      8      [ESC] to exit
|HR|      COMPATIBLE COMPUTER COMPANY
|HR|      WEEKLY TRANSACTION REPORT
|HR|      S      E
|HR|
|HR| Date      Company      Computer Units      Purchase Total
|HR| -----
|D| S      E S      E S      E S      E
|FR|
|FR|
Weekly Total: S      E

```

Press [ESC]. Your report is now marked.

From the marking menu, choose *Quit*.

## Other Report Writer Options

You can do much more with a report than you have done with this one. For example, you can:

- Mark the page to group companies and figures for subtotals
- Make a special title page
- Set the page length so you can print special forms, checks, or mailing labels.

## Leaving the Report Writer

R:base defaults will print this report in a standard 8.5 by 11 inch format, and since it is a brief report, you do not need to choose more options, so choose *Quit* from the menu. R:base displays this screen:

```

--Save changes--Discard changes--Return-----

```

You want to keep this report, so choose *Save changes*. R:base saves your report and brings an R> prompt on the screen.

## USING YOUR REPORT

### Screen Display

Now that you have created a report, try it out. You can display it on the screen with this command:

```
PRINT wklyrpt WHERE custid = 104
```

The WHERE clause limits the selection of rows for the report. Omit the clause if you want to print data from all rows in *transx*.

### Printed Report

If you have a printer attached to your computer, you can send the report to the printer with these two commands:

```
OUTPUT PRINTER
```

```
PRINT wklyrpt WHERE custid = 104
```

The OUTPUT command in this example instructs R:base to send the report to the printer. Your printed report will look similar to figure 3-3. Notice that your report displays the current date, not 03/15/85.

COMPATIBLE COMPUTER COMPANY WEEKLY TRANSACTION REPORT 03/15/85				
Date	Company	Computer	Units	Purchase Total
03/11/85	Industrial Concepts Inc.	CX3020	12	\$30,900.00
03/11/85	Industrial Concepts Inc.	MX3020	12	\$27,600.00
03/11/85	Industrial Concepts Inc.	PB3050	12	\$38,100.00
03/12/85	Industrial Concepts Inc.	PB3040	25	\$56,250.00
03/13/85	Industrial Concepts Inc.	MB3000	25	\$45,000.00
03/13/85	Industrial Concepts Inc.	CX3000	10	\$18,000.00
03/13/85	Industrial Concepts Inc.	PB3040	10	\$22,500.00
Weekly Total:				\$238,350.00

Figure 3-3 The Printed Report

When you have printed the report, enter the following OUTPUT command to return further R:base output to your screen:

```
OUTPUT SCREEN
```

## File Report

You can also send your report to a file. To do this, you use the OUTPUT command to name the file where the report is to go and then the PRINT command. Name the file *myreport.dat* and enter the commands like this:

```
OUTPUT myreport.dat  
PRINT wklyrpt WHERE custid = 104  
OUTPUT SCREEN
```

Executing the first two commands will place the report file on the default drive and the current directory. If you want to send it elsewhere, include the drive or directory.

You can see the report you stored by using the TYPE command. To display the file, at the R> prompt, type:

```
TYPE myreport.dat
```

Again, if the file is not on your default drive and current directory, include the drive or directory.

## MAINTAINING YOUR DATABASE

R:base offers you a complete set of tools for maintaining your data and your database structure. This section shows a few of the many commands available to you.

## Editing Data

The EDIT command allows you to change data you have stored in a table. Suppose, for example, that sales representative Thomas Green moves from Bellevue to Redmond. His new address is:

5500 Churchill St.  
Redmond, WA 98052

In the *salesrep* table, you need to change information in his row in three columns: *address*, *city*, and *zip*. Here is how you do it.

At the R> prompt, enter the EDIT command like this:

```
EDIT ALL FROM salesrep WHERE lastname = Green
```

R:base displays the row for *Green*. The first column, *empid*, is highlighted. Use the tab key to move the lighted block to the *address* column. Type in the new address.

Use the tab key to move to *city*. Note that R:base informs you that more data exists to the right of your current screen display. Also, tabbing to that field causes the screen to shift and to display those columns.

Type in the name of the new city.

Now, move to *zip* and type in the new zip code.

You have made the changes, so press [ESC] to exit.

R:base stores the new data. To see the updated table, use the SELECT command.

When you want to change data in more than one row, you can enter the command like this:

```
EDIT ALL FROM salesrep
```

R:base makes the whole table available for editing. You can move down through the rows with the up-arrow and down-arrow cursor keys. You can move from column to column by using the tab key to move right and the shift key and tab key together to move left.

When you finish, press [ESC] to exit.

For a table like *salesrep* that has a form, you can also use the form for editing. Enter the command like this:

```
EDIT USING repform WHERE lastname = Green
```

Using a table form in the EDIT mode enables you to view one row of the table at a time.





When the forms prompt menu comes up, choose *EDIT*. Try changing a few items. When you finish editing, press [ESC]. Then choose [Q] to quit.

### Adding a Column to a Table

You can add a column to a table with the *EXPAND* command. If it is a column that is not in the database, you specify the data type and, if *TEXT*, the length.

For example, suppose you decide to add a new column to the telephone-list table, *cstphone*. This column will be for an alternative telephone number for companies that have more than one. Name the column *altphone*.

At the *R>* prompt, enter the command like this:

```
EXPAND cstphone WITH altphone TEXT 12
```

Use the *SELECT* command to view the results. The table will look like this on your screen:

```
R> EXPAND cstphone WITH altphone TEXT 12
R> SELECT ALL FROM cstphone
```

company	phone	altphone
Computer Distributors Inc.	617-423-8921	
Computer Mountain Inc.	303-271-1500	
Computer Warehouse	812-701-1002	
Industrial Computers Inc.	303-239-7823	
Industrial Concepts Inc.	415-878-5600	
Midtown Computer Co.	312-277-5000	
PC Consultation and Design	415-892-6745	
PC Distribution Inc.	617-341-2189	
Southwest Computers, Inc.	512-581-8800	

```
R>
```

In the new column, *R:base* has supplied nulls (-0-), indicating that no value is present. You can replace the nulls with telephone numbers by using the *EDIT* command.

## Deleting Rows

When you want to remove one or more rows from a table, use the `DELETE` command. The command includes a `WHERE` clause that allows you to specify one or more rows from the table.

Suppose Thomas Green resigns from COMPUCO. You want to remove all data about him from the *salesrep* table. To do it enter the command like this:

```
DELETE ROWS FROM salesrep WHERE empid = 200
```

R:base removes the row. To see the results, use the `SELECT` command to display the table.

## Removing Tables

It is a good idea to do housekeeping on your database occasionally. One of the commands that is helpful for cleaning out data you no longer need is the `REMOVE` command. This command removes a table and its data from the database.

For example, in the COMPUCO database, you might prefer removing a table like *cstphone*, updating the data in the *custlist* table, and then making a new table for the telephone list.

To remove the table, enter the command like this:

```
REMOVE cstphone
```

R:base will remove the table.

Another table that can be removed is *templ*, the table you created with the `INTERSECT` command, since it duplicates data from two other tables.

## LEAVING R:BASE

When you finish working with this tutorial, remember to exit properly from R:base in one of these two ways:

- At the `R>` prompt, press `[ESC]`. Pressing `[ESC]` returns you to the R:base main menu.
- At the `R>` prompt, type `EXIT` and press `[ENTER]`. The `EXIT` command returns you to the R:base 5000 main menu.

## THE NEXT STEPS

By working your way through this tutorial, you have learned the basics for using R:base 5000. You can define a database, load data, make forms and reports, edit your data, manipulate tables, and maintain your database structure.

This should allow you:

To go to work and start using R:base. As you use R:base 5000, you will learn more about it. Read parts 2 and 3 of this manual for information about commands and functions not covered in this tutorial.

Or to go on immediately to the advanced tutorial in the next chapter and learn:

- How easily you can make an application with R:base 5000.
- How an application makes it easier to use your database.

---

## Advanced Tutorial Contents

<b>How to Use This Chapter</b>	4-2
<b>Getting Started</b>	4-2
Naming the Main Menu	4-4
Defining the First Menu Option: Sales Representative Data	4-5
Testing the Application	4-9
<b>Building the Application</b>	4-10
Defining the Second Menu Option: Customer Data	4-10
Getting Help from the EXPRESS	4-12
Defining the Third Menu Option: Product Data	4-12
Defining the Fourth Menu Option: Print Reports	4-13
Defining the Sales Representative Report	4-14
Defining the Customer and Product Reports	4-15
Installing a Custom Report	4-16
Defining the Fifth Menu Option: Customer Purchase History	4-16
Designing a Custom Macro	4-17
Building a Custom Macro	4-24
Installing a Custom Macro	4-25
<b>The Next Steps</b>	4-25

## HOW TO USE THIS CHAPTER

This chapter shows you how to use R:base 5000 for building and using applications.

Work your way through the entire chapter. When you have finished, you will know how to build application menus that integrate the database with R:base commands.

It is recommended that you become familiar with the database and application design for the sample *compuco* database presented in chapter 2 and with the basic R:base 5000 functions and commands described in chapter 3.

## GETTING STARTED

Be sure you have the *compuco* database files on the default drive (and current directory). See chapter 1 for more information.

As you begin building applications, keep in mind the difference between a database and a database application. A database is just a collection of tables of data. By itself, a database is cumbersome to use. A database application is a database plus menus and commands to process the database. These menus and commands provide a way for the user to keep data current, to print reports, and to answer questions from data.

Over the years, database experts have learned that, no matter how good they are, or how many years of experience they have, it is nearly impossible to build a database application correctly the first time. They have found that they need to start with something, get it working, add a little, take a little away, make other adjustments, and so forth, until gradually the database application takes shape. Most likely the same will be true for you. Happily, both R:base and the Application EXPRESS were designed with this in mind.

Building an application can seem overwhelming at first. As with other projects, the best strategy is to take it one step at a time. First, define the database with the EXPRESS. Then, design your menus, on paper. Next, use the EXPRESS to construct commands to define and process each menu. At each step, make adjustments when necessary. If you leave out a column, go back to the EXPRESS and add it. If you leave out a menu choice, go back to the EXPRESS and add it. Consider this iterative process for the *compuco* database.

Figure 4-1 shows the *compuco* application menu structure designed in chapter 2. You can use the EXPRESS to implement that structure in several ways. In this chapter, one method is described. Other ways of using the EXPRESS are explained in chapter 14.

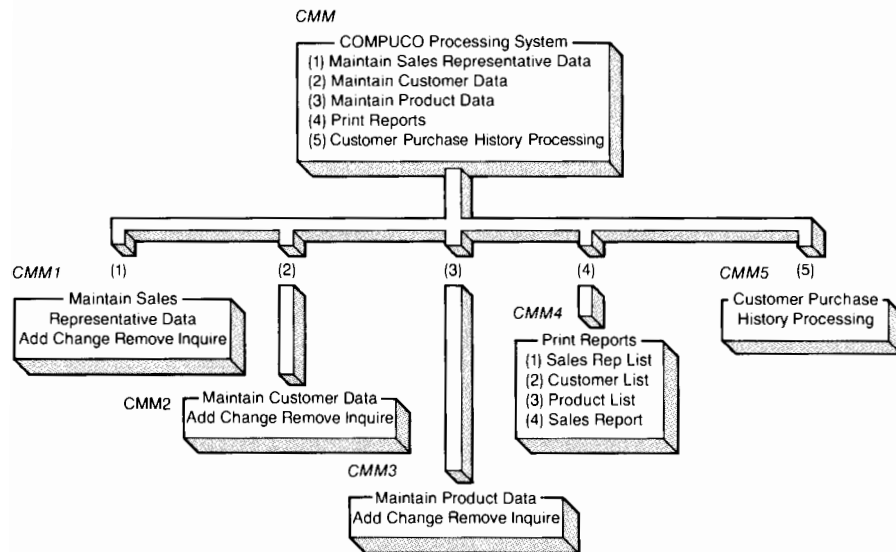
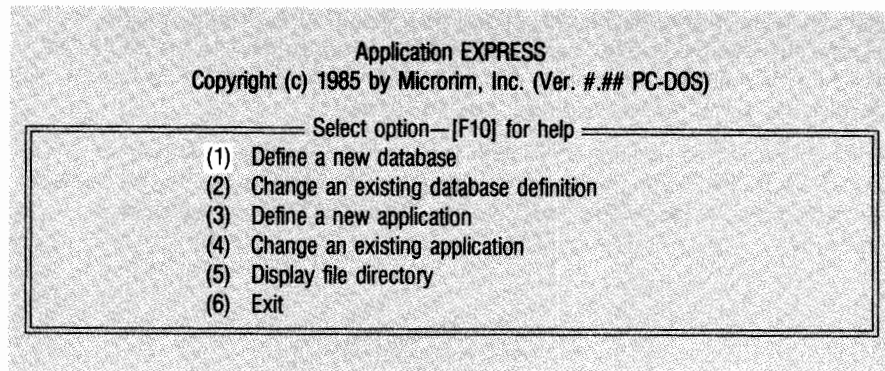


Figure 4-1 COMPUCO Application Menu Tree

First, start R:base 5000 and select *EXPRESS* from the first menu. The *EXPRESS* menu appears as shown below.



### **Naming the Main Menu**

Select option 3 to start the menu building process. The *EXPRESS* displays your current database names and asks you to select the one you want to process. Select *compuco* from the list.

Next, the *EXPRESS* asks you for an application name. The choice of name is entirely up to you. *Comp1*, for the first *compuco* application, is a good choice.

The *EXPRESS* then asks you to provide a name for the main menu. There are several ways of naming menus. Over time you can devise a scheme you particularly like. In figure 4-1, the main menu is called *CMM*, the menu corresponding to the first option is called *CMM1*, that for the next option is *CMM2*, and so forth. If you want to stay with this convention, enter *CMM* now. Otherwise, enter a name you like.

Next, the *EXPRESS* asks if you want the menu choices displayed horizontally or vertically. Figure 4-1 shows them vertically, so select that choice. Now the *EXPRESS* asks for a title to display when it shows the menu to the user.

Type:

COMPUCO PROCESSING SYSTEM

After entering the title, the EXPRESS automatically centers it on the menu border. Your screen looks like this:

Enter or change the menu choices— [ESC] when done

(1)

COMPUCO PROCESSING SYSTEM

[F1] Insert    [F2] Delete    [F3] Review    [F4] Reset value    [F5] Help  
Database COMPUCO   - Defining Application COMP1   - Menu CMM

### Defining the First Menu Option: Sales Representative Data

Now enter the text for the menu choices. You need enter only the first one for now. You can add the others later. Type:

MAINTAIN SALES REPRESENTATIVE DATA

Then press [ESC].



At this point, the EXPRESS wants to know if the operator may use the [ESC] key to exit from the menu. Except in very special circumstances, this action should be allowed, so select *Yes*. Next, the EXPRESS needs to know if you want to provide help text along with this screen. If so, the text you type will be displayed when the operator presses [F10]. You may want to provide such text, but you can always add it later. Select *No* for now. Your screen appears as follows:

COMPUCO PROCESSING SYSTEM

(1) MAINTAIN SALES REPRESENTATIVE DATA

The actions for menu selection 1 will be defined

Select action for this menu selection

Load Edit Delete Browse Select Print Custom Macro Menu Exit

[F3] Review [F5] Reset Value [F10] Help

Database COMPUCO - Defining Application COMP1 - Menu CMM

The EXPRESS gives you a number of options for defining this menu choice. Some of these are explained in this chapter. If you are curious now about the other options, look them up in chapter 14. In accordance with the structure shown in figure 4-1, select *Menu* from the list of actions and name the menu *CMM1*.

Now the EXPRESS asks you the same series of questions to define menu *CMM1*. Select vertical style and enter the menu title:

MAINTAIN SALES REPRESENTATIVE DATA

Now you are asked to list the menu choices. This time, enter them all. Use the choices *ADD*, *CHANGE*, *REMOVE*, and *INQUIRE*. Your screen appears as follows:

Enter or change the menu choices—[ESC] when done

MAINTAIN SALES REPRESENTATIVE DATA	
(1)	ADD
(2)	CHANGE
(3)	REMOVE
(4)	INQUIRE

[F1] Insert      [F2] Delete      [F3] Review      [F4] Reset value      [F5] Help  
 Database COMPUCO   - Defining Application COMP1   - Menu CMM1

When you press [ESC], the EXPRESS takes you back to menu *CMM* and asks if you want to do anything else after menu *CMM1* is processed. You do not, so select *No*.

Now define the actions for the choices in menu *CMM1*. First, select *Yes* for use of the [ESC] key and *No* for help text. (Remember, you can add it later if you want.) Now the EXPRESS asks you for actions for option 1, *ADD*. *LOAD* is the R:base command that adds records to a table. Therefore, choose *Load* from the list of actions. The EXPRESS asks for the table to load. Since you are adding sales representative data, enter *salesrep*.

If the database you are working with already has at least one defined form, EXPRESS displays the existing form names and adds the choice (*NEW*). If no forms are displayed, EXPRESS asks you to assign a name to the form. If your database has no defined forms, proceed with the instructions in the next paragraph. If *compuco* contains defined forms, select (*NEW*) from the menu because you are defining a new form.

You are now asked to assign a name to the form. Again, the choice of names is up to you. *SRI*, for *salesrep* form number 1, is a good choice. Now, you need to tell the EXPRESS which columns to load. Since you want to obtain all of the data, select *ALL*. Enter a title for the form. Type:

#### SALES REPRESENTATIVE DATA

Now enter text that you want displayed to prompt the user for data. By default, the EXPRESS displays the name of the column. You can use that, or you can substitute a more descriptive prompt message such as *ENTER EMPLOYEE ID*. The EXPRESS leads you through all the columns of the table.

Once you have defined form *SRI*, the EXPRESS asks if you want to do something else for this menu choice. Select *No*, and you can now define actions for option 2, *CHANGE*. EDIT is the R:base command for changing the rows in a table. Therefore, choose *Edit* from the list of actions. The EXPRESS next asks which table you want to edit. Enter *salesrep*. Next, you need to tell the EXPRESS which form to use. As before, you can define a new form but for now, form *SRI* is just fine. Therefore, select *SRI*.

You may want to sort data before editing it. For example, you may want to edit *salesrep* data in increasing order of *empid*. If so, you can tell the EXPRESS to do this. For now, ignore this option and press [ESC].

Now the EXPRESS asks if you want to select rows for editing by the value of a column. In other words, the EXPRESS asks the operator to provide a column value, and then locates all of the rows having that value. Only those rows will be edited. For *compuco*, the operator should enter a value for *empid*, and then edit only the rows in which *empid* equals that value. Therefore, select *empid*. Next, select *EQ*, since you want rows equal to the value the user enters. Then select *Yes*, since you want the user to provide a value, and finally, enter a prompt message such as:

#### ENTER EMPID OF THE RECORD TO CHANGE:

Now the EXPRESS asks if you want more actions taken for option 2. Select *No*.

For option 3, *REMOVE*, select *Delete* from the list of commands. The EXPRESS now asks you to describe how the operator will identify the rows to be deleted. The process is the same as that just described for editing. You need to provide the name of the table from which rows will be deleted (*salesrep*, in this case), the name of the column to identify the row to be deleted (*empid*), the comparison operator (in most cases, *EQ*), whether or not the operator will provide a value to a prompt (*Yes*), and a prompt message (ENTER EMPID OF THE RECORD TO REMOVE:).

Now define an action for the option 4, *INQUIRE*. When the application is finished and the operator selects this choice, rows of the table *salesrep* are displayed. In chapter 3, you learned how to do this with *SELECT*. Therefore, choose *Select* from the list of actions, and then choose *salesrep* from the list of tables. Next, you need to tell the EXPRESS which columns to display. You could choose the *ALL* option, but if you do, you will have a problem: only four or five columns can fit on a screen at one time. Therefore, choose some of the columns to display—for example, *empid*, *lastname*, *frstname*, and *phone*. Then, you can choose a column value for sorting. For now, ignore this option and enter [ESC]. Finally, you can let the user provide a value of a column for query purposes. Select *lastname*, then *EQ*, and then *Yes* to indicate the value of *lastname* will be entered by the operator. Finally, provide a prompt message such as:

ENTER SALESREPS LAST NAME:

At this point, you have defined all of the processing choices for the first choice of the main menu, *Maintain Sales Representative Data*. You are asked if you have additions to the menu. Select *No*. The EXPRESS now writes a large number of R:base commands to process your application. The message, *Writing application files—Please wait*, and a long list of commands appear on your screen. After all of the commands are generated, the EXPRESS tells you to press any key to continue.

The EXPRESS then asks if you want it to build a start-up file. Select *No* for this sample application. If you answer *Yes*, the EXPRESS creates a start-up file (*rbase.dat*) for R:base. Then, whenever you run R:base, the application you just constructed is automatically started. The presence of the file makes it difficult at this point of application development to get back to R:base without running the application.

Before defining more choices, try using the application you have just defined so you understand better what you have done.

## Testing the Application

To run the application you have defined, first leave the EXPRESS and enter R:base. Press [6] to exit from the EXPRESS menu, and then choose the RBASE option from the R:base 5000 main menu. Next, choose option 1, *R:base command mode*.

The application you have developed is called *comp1*, and it is stored in a file named *comp1.apx*. To run your application, at the R> prompt type:

```
RUN comp1 IN comp1.apx
```

Your application now appears on your screen. R:base displays *CMM*, the menu you defined. Now, select option 1 (the only one you have defined so far), and watch. Your menu *CMM1* appears on your screen. You are now given a chance to maintain the sales representative data. You can add, change, remove, and inquire about your data. To see how it all fits together, add a row, change it, inquire about it, and then delete it. Do this now before you go on.

## **BUILDING THE APPLICATION**

### **Defining the Second Menu Option: Customer Data**

Figure 4-1 shows the full menu tree for the *compuco* application. Now that you have defined *CMM1*, the next step is to define processing for menu *CMM2*. This menu is similar to *CMM1*, and is easy to define.

The EXPRESS is very forgiving. If you become completely lost, keep pressing [ESC] until you get back to the main EXPRESS menu, and then select *Change an existing application* to start over.

Type [ESC] to exit your application, and then type EXIT to leave R:base. Select *EXPRESS* to reenter the EXPRESS program. Select option 4, *Change an existing application* from the EXPRESS menu.

The EXPRESS then asks for the application you want to change. Choose *COMPI*. Next, the EXPRESS asks if you want to change the name of the application. Answer *No* (unless you want to change the name), and then select *Change menu text and actions* from the next menu. At this point, the EXPRESS asks what menu you want to change. Select *CMM*, since you are going to add a new choice to this menu.

At this point, the EXPRESS displays menu *CMM* and gives you a chance to change the title. Press [ENTER] unless you want to change the title. Next, the EXPRESS will ask if you want to change the wording in option 1. If you are happy with that wording, press [ENTER] again. Otherwise change the wording and then press [ENTER]. Now the EXPRESS displays 2 and waits. Here is your chance to add the new menu option. Type:

**MAINTAIN CUSTOMER DATA**

Press [ESC] since you are not going to define option 3 at this time. The EXPRESS now asks if you want to change the *escape* or *help* actions. Enter *No* unless you do.

(If you are feeling especially confident about your progress, you may want to define *help* now just to see how that is done. If so, answer *Yes* to define *help* and the EXPRESS will lead you from there. Once the *help* text is defined, continue with the next paragraph.)

Next, the EXPRESS asks if you want to change the option actions. Choose *Yes* since you need to define actions for the new option 2. If you want to review the choices, you can answer *Yes* to the next question and the EXPRESS will review the processing for you. For now, enter *No*, and then type 2 to indicate that you want to change the action for option 2. The EXPRESS indicates that no action is defined for this option and asks you to press [ENTER] to define actions.

Press [ENTER] and you can begin defining the actions for maintaining customer data. The process is identical to the process used for defining menu *CMM1*. Try to work through this process by yourself. In case you get stuck, the following is a script of responses you can use to define the first two choices on menu *CMM2*. You can do the rest by yourself.

1. Choose *Menu*
2. Type *CMM2*
3. Choose *vertical*
4. Type *MAINTAIN CUSTOMER DATA*
5. Type the four choices *ADD, CHANGE, REMOVE, INQUIRE*, then press [ESC]
6. Choose *No* more actions
7. Choose *Yes* for ESC
8. Choose *No* for HELP
9. Choose *Load* for first choice
10. Choose *custlist*
11. Choose *NEW* form
12. Type *Cust1* for form name
13. Choose *ALL*
14. Type *MAINTAIN CUSTOMER DATA*
15. Adjust prompts, if you want, then press [ESC]
16. Choose *No* more actions
17. Choose *Edit* for second *CMM2* choice
18. Choose *custlist*
19. Choose *cust1* (since you can use the same form for load and edit)
20. Choose [ESC] to indicate no more sorting
21. Choose *custid* for selection
22. Choose *EQ* for comparison
23. Choose *Yes* to enter value to edit at the time of execution
24. Type prompt message such as *Enter custid of row to change*
25. Select *No* more actions

At this point, the first two choices have been defined. Follow a similar process to define actions for the *REMOVE* and *INQUIRE* choices.



### **GETTING HELP FROM THE EXPRESS**

As you add more menus and choices to your application, the number of names, actions, and other items you must remember increases. The EXPRESS has a feature to help aid your memory. When you are defining actions for a menu option, you can press [F3] to display a help screen which shows the structure of the menus you have developed. This structure appears in outline form with the main menu first, the submenus next, and the sub-submenus below (there are no sub-submenus in this example application.)

The EXPRESS also positions the reverse video bar on the menu you are currently working on. If you move the bar to another menu and press [F3] again, the EXPRESS summarizes the options and actions for that menu. This feature is useful for keeping track of your application structure. Use it as you begin to develop more complex applications.

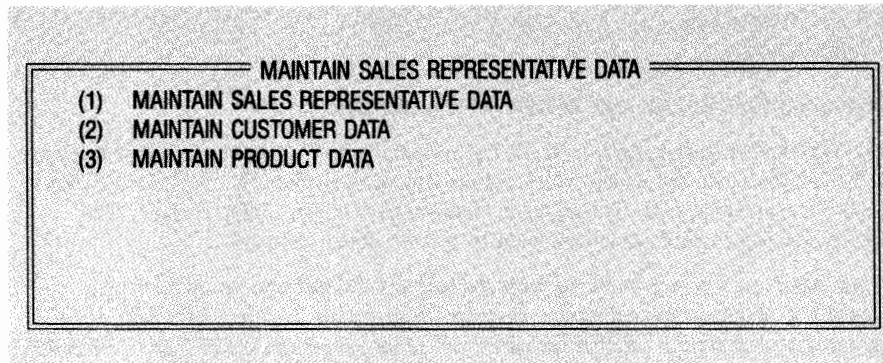
### **Defining the Third Menu Option: Product Data**

The next step in building the *compuco* application is to define menu option 3, *Maintain Product Data*. The process is similar to that used for options 1 and 2, so it will not be repeated here.

Once you have defined those three menu choices, enter R:base and run your application. From this you will get a feel of how the application will operate.

## Defining the Fourth Menu Option: Print Reports

If you have been following the instructions in this chapter, your main menu (*CMM*) currently has three choices as shown below:



The next step is to add a fourth choice for printing reports. To do this, enter the EXPRESS, choose option 4, *Change an existing application*, and select *COMP1* as the application name. When the EXPRESS asks if you want to change the name of the application, answer *No* (unless you want to change the name), and then select *Change menu text and actions* from the next menu. Select *CMM* from the menu and define menu option 4, *Print Reports*. Answer *No* because you do not want to change [ESC] and HELP actions. Answer *Yes* to change the pick actions, and *No* because you do not want to review all pick actions. Select pick 4, and press [ENTER] to define actions.



Select *Menu* as the action to perform, and name the new menu *CMM4*. Choose the menu type (vertical or horizontal) and enter the menu title, *Print Reports*. Define menu *CMM4* as having four choices:

- Print Sales Representative List
- Print Customer List
- Print Product List
- Print Report

#### **DEFINING THE SALES REPRESENTATIVE REPORT**

The EXPRESS asks questions about the menu processing you want. Answer these as before until you are asked which action you want performed for option 1, *Print Sales Representative List*. At this point, choose *Print* as the action to take. The EXPRESS asks which table you want to print. Choose *salesrep*.

If the database you are working with already has at least one defined report, EXPRESS displays the existing report names and adds the choice (*NEW*). If no reports are displayed, EXPRESS asks you to assign a name to the report. If your database has no defined reports, proceed with the instructions in the next paragraph. If *compuco* contains defined reports, select (*NEW*) from the menu because you are defining a new report and skip the following paragraph.

You are then asked to name the report to print. Enter a report name; *sr-rpt* is a good choice. Then you are asked to choose between column-wise and row-wise orientation. Column-wise orientation lists the column headings at the top of the report and prints the information row by row. Row-wise orientation lists the headings along the left side of the report and prints each row item with its corresponding heading. Choose column-wise orientation.

Now, the EXPRESS asks what columns you want printed on the report. Your choices here depend on the purpose of the report. Assume this report is to be used as a telephone list. If so, choose columns *lastname*, *firstname*, *phone*, and *ext*. Pick the names in that order since the order you choose will be the order in which the columns appear on the report. Next, the EXPRESS asks for a title to print at the top of the report. You can pick your own title; *Compuco Sales Representatives* or a similar title works well. Now you are asked to enter column headings (report labels) for your report. The EXPRESS will suggest the column name in the database. You can substitute other names if you want.

At this point, the EXPRESS will give you the option of sorting the data before it is printed. Usually telephone lists are in alphabetical order by last name. Therefore, choose *lastname* and then *firstname*. (This means that *firstname* will be used as a tie breaker if two sales representatives have the same last name.) Press [ESC] to leave the sort selection.

The EXPRESS now asks if you want to select specific rows for printing. In this case, you probably want all of the sales representatives, so you do not want rows qualified before printing. Press [ESC]. Answer *No* when the EXPRESS asks if you want another action for that option.

### DEFINING CUSTOMER AND PRODUCT REPORTS

You can define reports for *customer* and *product* tables in a similar fashion. Suppose that for the customer report, you only want customers that reside in a particular zip code area. In that case, choose *Print* from the list of actions for option 2, *PRINT CUSTOMER LIST*. Then choose *custlist* as the table to print, and choose *NEW* to define a new report. *cust-rpt* is a good choice for a report name.

Assume this report is to show company, customer number, first and last names, and zip code. Thus, pick *company*, *custid*, *firstname*, *lastname*, and *zip*. Again, choose the columns in the order you want them to appear on the report. Press [ESC] and enter *CUSTOMERS BY ZIP CODE* as the report title. Then enter the column titles.

Next, the EXPRESS asks for sorting criteria. Again, the choice depends on the purpose of the report. For this report, choose *company*, and press [ESC]. At this point, you can define selection criteria. Choose *zip* since you want to print only customers in a particular ZIP code. Next choose *EQ* since you want customers in the ZIP code you provide. (Observe that *greater than (GT)* or *less than (LT)* could also be useful choices to identify customers in particular geographic areas.) The EXPRESS next needs to know if the value is to be provided at execution time (that is, when the operator chooses to print the report). Answer *Yes* and then, for the prompt message, type:

ENTER ZIP CODE FOR CUSTOMERS YOU WANT:

Your report is now completed.

Before defining the remaining reports, you may wish to print the reports you have already defined. If so, save the changes you have made by pressing [ESC] until you are prompted to *Save* or *Abort*. Select *Save*. Then exit the EXPRESS, enter R:base, and at the R> prompt type:

RUN comp1 IN comp1.apx

Choose option 4 from the main menu, and then options 1 or 2 to print either of the reports.

Defining the *Product List* report is similar to the process described for *sr-rpt* and *cust-rpt*. You can follow the process on your own.

## INSTALLING A CUSTOM REPORT

The reports that are prepared by the EXPRESS are generalized, generic reports. You may not like the format or you may want choices for your reports beyond what the EXPRESS allows. If so, you can use the report writer in R:base. REPORTS provides a comprehensive set of features for producing customized reports that can be tailored to a wide variety of situations.

The introductory tutorial shows you how to prepare customized reports and chapter 11 covers the subject in complete detail. This section explains how to install a custom report into a menu using the EXPRESS.

The version of *compuco* that comes with R:base 5000 includes a predefined custom report named *salesrpt*. Suppose you want to install this report into option 4 of menu *CMM4*. To do that, enter *Change an existing application* from the main EXPRESS menu. Choose application *COMP1*, answer *No* instead of changing the name, and select *Change*. Then select menu *CMM4*. Next, choose option 4 to change, and select *Print* as the action to take. At this point, the EXPRESS asks for the table to print. Select *transx* since *salesrpt* is based on this table. Now select *salesrpt* from the list of reports. (The EXPRESS knows this table already exists because it has examined the *compuco* database and found it.) You are now given the choice of sorting or selection criteria. For now ignore these options and press [ESC] for both of these choices. At this point, you have installed *salesrpt* into your application.

## Defining the Fifth Option: Customer Purchase History

The advantage of using the EXPRESS is that you can quickly and easily develop menus and the commands to process them. The disadvantage is that only certain standardized processes are available. You may, however, have a need that is not readily satisfied with one of the standard options of the EXPRESS.

R:base has many commands for customized processing, which are explained in parts 2 and 3 of this manual. Using these commands, you can build macros that process the database in almost any manner you can envision. Those macros can stand on their own; they need not be integrated with an application constructed with the EXPRESS.

It is also possible to have the best of both worlds. You can use the EXPRESS to develop applications for processing standard requests, and write your own macros (or programs) for custom processing. Once you have written these macros, you can install them into an application using the EXPRESS. To illustrate this process, consider the following example.

**DESIGNING A CUSTOM MACRO**

Suppose that the COMPUCO customer service department needs to know customer purchase histories. Specifically, when a customer calls this department, a clerk asks the customer for his or her last name. Then, the clerk uses this name to access the *compuco* database to determine the items the customer has purchased.

If you examine the *compuco* database, you will see a problem: the data required to meet this need resides in three tables. Customer last names are stored in the *custlist* table. Customer purchases are stored in the *transx* table, and product descriptions are stored in the *product* table.

To obtain a customer purchase history, you need to access the three tables in the following manner:

1. Access *custlist* by the last name and obtain the value of the customer's id number (the value in column *custid*).
2. Look in the *transx* table, and find all of the rows having that value of *custid*.

This gives a list of items the customer has purchased. Unfortunately, this will be a list of product identification numbers. To obtain the product names, you need to look up each of the selected product numbers in the *product* table and obtain the corresponding value of *prodname*.

Does this sound complicated? It probably does, and computer experts have learned that processes like these (sometimes called algorithms) can be better understood if written in a stylized form of English called pseudocode (or false code). As an example, here is the pseudocode for the algorithm to produce customer purchase histories:

```
TURN OFF ERROR MESSAGES AND WARNING MESSAGES
OBTAIN CUSTOMER'S LAST NAME
LOOK FOR CUSTOMER'S LAST NAME IN CUSTLIST
IF THE CUSTOMER NAME IS IN THE TABLE THEN
    SHOW ON SCREEN: CUSTID NAME PHONE FROM THE CUSTLIST TABLE
ELSE
    PRINT AN ERROR MESSAGE
    REPEAT REQUEST FOR CUSTOMER'S LAST NAME
END OF IF
OBTAIN CUSTOMER'S ID NUMBER
LOOK IN THE TRANSX TABLE
IF THE CUSTOMER HAS NO TRANSACTIONS THEN
    PRINT AN ERROR MESSAGE
END OF IF
FOR EACH ROW HAVING A VALUE OF CUSTID = THE CUSTOMER'S ID
    OBTAIN THE VALUE OF PRODID
    LOOK UP PRODNAME IN PRODUCT FOR THAT VALUE OF PRODID
    IF PRODID DOES NOT EXIST IN BOTH TRANSX AND PRODUCT THEN
        PRINT AN ERROR MESSAGE
    END OF IF
    SHOW THE VALUE OF LASTNAME AND PRODNAME ON THE SCREEN
REPEAT FOR THE NEXT ROW IN TRANSX FOR THIS CUSTOMER
PROMPT FOR ANOTHER CUSTOMER ID NUMBER
TURN ON ERROR MESSAGES AND WARNING MESSAGES
```

The following sequence of R:base commands implements the logic shown above. The first line is a comment that lists the title of the file. SET VARIABLE *lnam* establishes a variable that is used to locate customer information. SET ERROR VARIABLE creates a variable, *ev*, that records an error that occurs during processing. The following two commands direct R:base not to display error or standard messages. SET VARIABLE *curpos* establishes a variable that is used to place the cursor. The next two commands, SET VARIABLE *loop* and WHILE *loop*, establish a processing loop so that continuous requests can be entered. (Be patient. This will become clearer as you read on.)

```
*(CMACRO1)
SET VARIABLE Inam TEXT ; SET ERROR VARIABLE ev
SET ERROR MESSAGE OFF ; SET MESSAGE OFF
SET VARIABLE curpos TO " "
SET VARIABLE loop TO 0
WHILE loop EQ 0 THEN
  NEWPAGE
  WRITE "CUSTOMER PURCHASE HISTORY" AT 2 28
  FILLIN Inam USING "TYPE THE CUSTOMER'S LAST NAME OR PRESS [ENTER] TO +
  QUIT: " AT 4 1
  IF Inam FAILS THEN
    BREAK
  ENDIF
  SHOW VARIABLE curpos AT 9 1
  SELECT custid lastname firstname phone FROM custlist +
  WHERE lastname = .Inam
  SET VARIABLE erresult TO .ev
  IF erresult NE 0 THEN
    NEWPAGE
    WRITE "TRY AGAIN, FOLLOWING LAST NAME NOT FOUND:" AT 5 1
    SHOW VARIABLE Inam AT 5 44
    GOTO continue
  ENDIF
  FILLIN id USING "TYPE THE CUSTOMER'S ID OR PRESS [ENTER] TO QUIT: " AT 6 1
  IF id FAILS THEN
    BREAK
  ENDIF
  SET POINTER #1 e1 FOR custlist WHERE custid = .id
  IF e1 NE 0 THEN
    NEWPAGE
    WRITE "CUSTOMER NOT IN OUR RECORDS" AT 8 27
    SHOW VARIABLE curpos AT 9 1
    GOTO continue
  ELSE
```

```
    SET VARIABLE ln TO lastname IN #1
ENDIF
    SET VARIABLE row TO 8
    SET POINTER #1 e1 FOR transx WHERE custid EQ .id
    IF e1 NE 0 THEN
        NEWPAGE
        WRITE "NO TRANSACTIONS EXIST FOR THIS CUSTOMER" AT 8 21
        SHOW VARIABLE curpos AT 9 1
        GOTO continue
    ENDIF
    NEWPAGE
    WRITE "CUSTOMER PURCHASE HISTORY" AT 2 28
    SHOW VARIABLE curpos AT 3 1
    SELECT custid frstname phone company=30 FROM custlist +
    WHERE custid = .id
    WHILE e1 EQ 0 THEN
        SET VARIABLE prod TO prodid IN #1
        SET POINTER #2 e2 FOR product WHERE prodid EQ .prod
        IF e2 NE 0 THEN
            NEWPAGE
            WRITE "ERROR - PRODUCT IN TRANSX BUT NOT IN PRODUCT" AT 9 18
            SHOW VARIABLE curpos AT 9 1
            GOTO continue
        ENDIF
        SET VARIABLE pname TO prodname IN #2
        SHOW VARIABLE ln AT .row 1
        SHOW VARIABLE pname=35 AT .row 20
        SET VARIABLE row TO .row + 1
    NEXT #1 e1
ENDWHILE
LABEL continue
SHOW VARIABLE curpos
FILLIN cont USING "                PRESS [ENTER] TO CONTINUE "
ENDWHILE
SET ERROR MESSAGE ON ; SET MESSAGE ON
NEWPAGE
RETURN
```

The next command, `NEWPAGE`, simply tells R:base to clear the screen, and the `WRITE` command tells R:base to write the phrase `CUSTOMER PURCHASE HISTORY` on the screen starting at row 2, column 28.

The command `FILLIN lnam` tells R:base to obtain a value from the user at the keyboard. Specifically, R:base is supposed to obtain a value for the variable `lnam`. The `USING` phrase means put whatever is inside the quotes (in this case, the phrase `TYPE THE CUSTOMER'S LAST NAME OR PRESS [ENTER] TO QUIT`) at row 4, column 1, and then wait for the user to enter a value to be placed in the variable `lnam`.

If the user presses `[ENTER]`, the following two lines direct the program to the final `RETURN` command and stop execution of the macro.

If the user enters a customer last name, the command `SHOW VARIABLE curpos` locates the cursor on the screen at row 9, column 1. If the name matches a name in `custlist`, the `SELECT` command displays the customer ID, name, and phone. Note that you can enter an asterisk instead of a name to list all names in the table. If the name is not valid, the `ERROR VARIABLE erresult` traps the error message and prevents the error from aborting execution. `NEWPAGE` clears the screen. The invalid name is displayed and a message asks the user to enter another name.

The command `FILLIN id` functions in the same way as the previous `FILLIN` command. If the user presses `[ENTER]`, the following two lines direct the program to the final `RETURN` command, which in turn causes the main menu to be displayed.

If the user enters a customer identification number, the `IF` statement is not true and `SET POINTER`, the command after the `ENDIF` statement, is executed.

The `SET POINTER` command is very important and useful. Pointers are used to process tables a row at a time. This command instructs R:base to set pointer number 1 to the first row in `custlist` in which column `custid` is equal to the value of the variable `id`. (Recall that `id` was assigned the value of the customer's identification number in the `FILLIN` command above.)

You may be wondering, why is there a period in front of `id`? The period tells R:base that `id` is a variable and not a value. If the period were not there, R:base would search `custid` for the two-letter entry `id`.

The `SET POINTER` command includes the variable `e1`. (Names like `curpos`, `loop`, and `e1`, by the way, are arbitrary. They could be `P7` or `NNN`, or some other name.) If there is a row in `custlist` for this value of `custid`, then `e1` is set equal to 0. If no rows satisfy the condition, `e1` is set to 2406.



The next command is an IF. When R:base receives this command, it does the following: if the variable *e1* equals 0, R:base executes the statements that follow the word ELSE. If *e1* is not equal to 0, R:base writes CUSTOMER NOT IN OUR RECORDS starting at row 8, column 27. The following command, SHOW VARIABLE *curpos*, returns the cursor to the beginning of the line. The command GOTO *continue* directs the system to find the command LABEL *continue* (seven lines from the end). The SHOW VARIABLE command adds a blank line beneath the message just displayed, and the FILLIN command displays the message PRESS [ENTER] TO CONTINUE. FILLIN retains the message on the screen until the user presses [ENTER] to continue.

If *e1* equaled 0 in the previous IF command, the SET VARIABLE command then tells R:base to set the value of variable *ln* equal to a value of column *lastname* in the *custlist* table. As you know, however, there are many values of *lastname* in this table. The row pointer (#1) tells R:base which one to pick. R:base is supposed to look in *custlist* and find a row that has a value of *lastname* equal to the value of *.id*. (Remember that *id* is the name of the variable that was set equal to the customer's ID number in the above command, SET POINTER #1.) Suppose the user entered 104 when prompted to enter the customer's ID number. If so, the value of variable *id* is now 104. When the SET VARIABLE command is executed, R:base will look in *custlist* for a row having a value of 104 in the *custid* column. It will then set the variable *id* equal to the *lastname* of that row.

The following command sets another variable, called *row*, to the value of 8. This variable controls where the output is placed on the screen. Output will start on line 8 and be placed on subsequent lines as the program progresses.

The command SET POINTER #1 is now used to retrieve information from the table *transx* where the column *custid* has a value equal to the value of the variable *id*.

Once again, IF processing is used to determine whether information pertaining to the ID number entered is available in the table *transx*. If information is not available (*e1* is not equal to 0), a message is displayed and the operator must press [ENTER] to continue. If information is available (*e1* equals 0), processing continues with the command that follows the ENDIF command.

If there is data in the *transx* table for the customer specified, NEWPAGE clears the screen, and WRITE displays the heading CUSTOMER PURCHASE HISTORY at row 2, column 28. The command SHOW VARIABLE *curpos* locates the cursor at row 3, column 1. SELECT displays the customer ID, first name, phone number, and company name information from *custlist*.

The WHILE/ENDWHILE loop enables R:base to locate each of the rows in *transx* that contain valid customer purchase information. In this case, all the commands between the word WHILE and the word ENDWHILE are processed repetitively until *e1* is given a non-zero value. Thus, these statements are executed as long as there remain rows in *transx* that correspond to the customer *id* being processed and as long as the product in question exists in both *transx* and *product*.

In the first statement on the WHILE loop, the variable *prod* is set to the value of *prodid* in the row identified by pointer number 1 (#1). Then, a new row pointer (#2) is established to determine if the product that exists in *transx* has a corresponding entry in *product*. If the product does not exist in *product*, an error message is displayed. (The existence of the product is determined by the statement IF *e2* NE 0.) The WHILE loop is exited and the operator is prompted to press [ENTER] to continue.

If the product in *transx* has a corresponding product entry, the variable *pname* is set to the value of *prodname* in the row of *product* that has *prodid* equal to value of *prod*. These statements obtain the name of the product that the customer has purchased. The results are then written to the screen. The customer's last name is displayed on row number *.row* (currently 10), column 1. The name of the product purchased is printed on the same row, column 20. The =35 included after *pname* adjusts the display width for the product to 35 characters. (The default variable display width is 30.)

The next statement adds one to the value of the variable *.row*. Thus, the next product name is displayed on row 9. Finally, the command NEXT #1 *e1* tells R:base to find the next row for pointer number 1. This is a row in *transx* having a value of *custid* equal to the value of *.id*. If there are no more rows for that customer ID, R:base sets *e1* to a non-zero value and terminates processing the inner WHILE loop.

The message PRESS ENTER TO CONTINUE is displayed. After the user presses [ENTER], the process is repeated for a new customer identification number. When the program is exited, error messages and R:base messages are set on, and NEWPAGE clears the screen.

This macro probably appears more complex than it is. Study the commands for a few minutes to be certain you understand what each of them does. These are fundamental commands for building custom macros, and if you understand them, you will be able to define a great deal of customized processing for yourself.

**BUILDING A CUSTOM MACRO**

There are two ways of constructing custom macros. One is to use the R:base 5000 editor, called RBEDIT, to type in the application commands and store them as a macro in a DOS file. From there the macro can be executed directly from R:base, or it can be installed into an application with the EXPRESS.

Another way is to use the EXPRESS to build the macro. If, when you are defining actions for processing a menu choice, you select the *Custom* option from the list of commands, the EXPRESS allows you to edit and then installs your macro into the application when you are finished.

For most cases, the first alternative is preferred. Build your macro with RBEDIT and store it in a DOS file. You can run it from R:base to make certain it is correct. Once you know your macro is complete, you can install it into the application using the EXPRESS.

To access the editor, enter R:base command mode, and, at the R> prompt, type:

**RBEDIT**

Select *New file*. You now have a blank screen on which to type commands. For practice, type in the macro shown above. If necessary, you can add lines using the [F1] key. This is useful if you forget to type a command. You can remove lines with the [F2] key. You can also add and delete characters with the [Ins] and [Del] keys.

Once you have typed the macro, press [ESC], and then select *Save*. RBEDIT will ask for file name. Name it *cmacro1* and press [ENTER]. Then press [Q] to exit RBEDIT, and you will be back at the R> prompt.

Try your new macro.

1. To open the database, type *OPEN compuco*

2. Type *RUN cmacro1*

If you typed your macro correctly, you will be prompted for a customer last name.

3. Type *James*

After you have entered James' id number, you will see a list of the products that James has purchased. Try *xxx* and you should get the error message that there are no records for that customer.

If your macro is not working correctly, reenter RBEDIT and fix it. Type RBEDIT at the R> prompt, and then select *Old file*. Type *cmacro1* when asked for the file name. RBEDIT will then display your macro, and you can correct it. Then exit RBEDIT and try your macro again. Follow this process until you have a correct macro.

### INSTALLING A CUSTOM MACRO

Once you have completed and checked your macro, you can install it in the *compuco* application. To do this, exit R:base and enter the EXPRESS. Choose option 4, *Change an existing application*, and then select the option to change menu *CMM*. You can now add the fifth (and final!) choice to this menu. After the 5, type *CUSTOMER PURCHASE HISTORY PROCESSING* and then answer the series of questions leading to defining the actions for option 5. At this point, select *Macro*. The EXPRESS asks for the name of your macro. Type *cmacro1*. You are then given a chance to edit your macro. Select *No* since you already know that it is correct. Now you can exit the EXPRESS. Your macro has been installed into the *compuco* application.

### THE NEXT STEPS

This chapter has introduced you to the process of building application menus with the EXPRESS. In the process, you have learned how to define basic forms and reports. You have also been introduced to custom programming with R:base 5000.

You should be able to define many applications with the knowledge that you have at this point. To take advantage of the full capabilities of R:base 5000, read parts 2 and 3 of this manual as you begin building more complex applications.



---

## R:base Fundamentals Contents



<b>How to Use this Chapter</b>	5-3
<b>The R:base Main Menu</b>	5-3
<b>Screen Prompts</b>	5-3
<b>Command Syntax</b>	5-4
R:base Commands	5-5
R:base Keywords	5-6
R:base Arguments	5-7
Entering R:base Commands	5-9
Command Comments	5-9
Abbreviating Commands	5-9
Entering Commands Longer Than 79 Characters	5-10
Entering More Than One Command or Data Row on a Line	5-10
Reentering the Previous Command	5-11
<b>Command Clauses</b>	5-13
Sorting Information with a SORTED BY Clause	5-13
Qualifying Information with a WHERE Clause	5-14
Entering More Than One WHERE Condition	5-16
The WHERE Clause and Key Processing	5-17
Entering SORTED BY and WHERE Clauses for the Same Command	5-17
<b>Database Commands</b>	5-18
Requesting On-Line Help with HELP	5-18
Requesting On-Line Prompts with PROMPT	5-20
Opening a Database with OPEN	5-22
Closing a Database with CLOSE	5-22
Exiting R:base with EXIT	5-23
Clearing the Screen with NEWPAGE	5-23
Identifying the Database User with USER	5-23
Identifying the Source of Incoming Information with INPUT	5-23
Identifying the Destination of Information with OUTPUT	5-24
Removing Unusable Space from the Disk with PACK	5-25
Setting System Default Values and Variables with SET	5-25
Special Characters	5-26
Setting Special Characters	5-26
Operating Conditions	5-27
Setting Keyword Values	5-27
Variables	5-30
Global Variables	5-31
Error Variables	5-31
Displaying System Default Values and Variables with SHOW	5-32

<b>Database Back Up and Recovery</b>	5-33
The UNLOAD Command	5-34
The RELOAD Command	5-36
<b>Operating System Commands</b>	5-36
<b>Database Files</b>	5-37
Contents	5-37
Estimating File Sizes	5-38
Estimating the Size of a Sort File	5-39

## HOW TO USE THIS CHAPTER

This chapter presents basic information on using R:base screen prompts and commands, backing up your database, and working with R:base files. If you are not familiar with R:base, read the entire chapter for a good understanding of these essentials. If you are familiar with R:base you can read only the sections you need for specific information.

The chapter begins with an explanation of the R:base main menu options and the eight screen prompts R:base uses to show the operating mode you are using. Most of the chapter is devoted to R:base commands: the syntax used for entering commands, the command clauses used to organize and locate specific information, the commands used for daily operation, the commands used to back up your database, and the available operating system commands. The final section describes database files and how to estimate their size.

## THE R:BASE MAIN MENU

The R:base main menu is displayed when you choose the *R:base* option from the R:base 5000 main menu. There are four options you can choose:

- *R:base command mode* directs you to an R> prompt, where you can immediately enter R:base commands.
- *HELP for using R:base* displays a task oriented menu. Choose the management task you want to familiarize yourself with and read the on-line assistance.
- *PROMPT mode for command assistance* displays the list of commands that have command assistance available. Enter a command name, and the PROMPT mode explains each part of the command line and helps you execute the command.
- *EXIT* returns you to the R:base 5000 main menu.

You can return to the main menu at any time by pressing [ESC] when an R> prompt is displayed.

## SCREEN PROMPTS

R:base uses eight screen prompts. The screen prompt is an uppercase letter followed by a greater-than (>) symbol. Table 5-1 shows each of the screen prompts, the corresponding R:base operating mode, the common prompt name, and the chapter where you can obtain further operating information.



Table 5-1 R:base Screen Prompts

Prompt	Mode Indicated	Common Name	Chapter
<b>D&gt;</b>	Database definition	Define	6
<b>H&gt;</b>	HELP command	Help	5
<b>I &gt;</b>	If-Then-Else processing	If	15
<b>L &gt;</b>	Data loading	Load	7
<b>P&gt;</b>	Command prompting	Prompt	5
<b>R&gt;</b>	R:base command module	R:base	5
<b>W&gt;</b>	While loop processing	While	15
<b>+ &gt;</b>	Continuation of previous line	Plus	5

## COMMAND SYNTAX

An R:base command statement instructs R:base to perform a specific function. R:base command statements consist of an R:base command, keyword, and one or more arguments. Each time you enter an R:base command statement, the commands and keywords for that command statement always remain the same. The arguments, however, vary depending upon the specific task you need to accomplish. For example, in the command

```
SELECT collist FROM reports
```

the R:base command *SELECT*, and the keyword *FROM*, never change. The keywords *collist* and *reports* are arguments that can change each time you enter the command line.

If you enter the command syntax incorrectly, R:base responds with an error message that displays the correct syntax.

## R:base Commands

Table 5-2 shows a complete list of R:base commands. Command names shown with an asterisk may not be used as column, table, or password names. The commands are fully explained in this manual and briefly explained in chapter 2 of the *R:base 5000 Reference Manual* "Command Dictionary."

Table 5-2 R:base Commands

APPEND	DELETE	GOTO	OWNER*	RULES*
ASSIGN	DIRECTORY	HELP*	PACK	RUN
BREAK	DISPLAY	IF	PASSWORDS*	SELECT
BUILD KEY	DRAW	INPUT	PAUSE	SET
CHANGE	EDIT	INTERSECT	PRINT	SET ERROR VARIABLE
CHANGE COLUMN	EDIT VARIABLE	JOIN	PROJECT	SET POINTER
CHDIR	ELSE	LABEL	PROMPT*	SET VARIABLE
CHDRV	END*	LIST*	QUIT	SHOW
CHECK	ENDIF	LOAD	RBEDIT	SHOW VARIABLE
CHKDSK	ENDWHILE	MKDIR	RELOAD	SUBTRACT
CHOOSE	ENTER	MOVE	REMOVE	TABLES*
CLEAR	ENTER VARIABLE	MPW*	REMOVE COLUMN	TALLY
CLOSE	ERASE	NEWPAGE*	RENAME	TYPE
COLUMNS*	EXIT	NEXT	RENAME FILE	UNION
COMPUTE	EXPAND	NOCHECK	REPORTS*	UNLOAD
COPY	FILL	NOFILL	RETURN	USER
DATE	FILLIN	OPEN	RMDIR	WHILE
DEFINE	FORMS*	OUTPUT	RPW*	WRITE

\*Commands you cannot use as column, table, or password names

**R:base Keywords**

Table 5-3 shows a complete list of R:base keywords and reserved terms. Entries shown with an asterisk may not be used as column, table, or password names.

Table 5-3 R:base Keywords and Reserved Terms

= A*	CYAN	GREEN	PAGE*	SYNTAX
ALL*	= D*	GRAY	#PAGE	TABLE
ALL=S*	DATA*	HEADING*	PAGESIZE*	TABLE1*
AND*	DATABASE*	HEADINGS*	PGDN	TABLE2*
AND/OR*	DATE	IN*	PGUP	TERMINAL*
Apr	#DATE	INTEGER	PLUS	TEXT
AS*	Dec	ITEM	POINTER	THEN*
ASCII	DELETE	Jan	PRINTER*	TIME*
AT*	DETAIL*	Jul	QUOTES	#TIME
ATT*	DIF	Jun	RDATA*	TO*
ATTRIBUTE*	DOLLAR	KEY*	REAL	USER
Aug	DUPLICATE*	KEYBOARD	RED	USES
AUTOSKIP	ECHO	LAST*	RELATION*	USING*
AVE*	END*	LAYOUT*	REORDER	VALUE
BACKGRND	ENDC*	LIGHT	REPEAT	VARIABLE
BEGINS*	ENDS*	LIMIT*	REPORT*	WHERE*
BELL	ENTER	LINES	RESET*	WHITE
BLACK	ERROR	MAGENTA	#RETURN	WIDTH
BLANK	ESC	Mar	REVERSE	WITH*
BLUE	ESCAPE	MAX*	RNAME*	YELLOW
BODY*	EXISTS*	May	ROW	-0-
BOOLEAN*	FAILS*	MESSAGES	ROWS*	
BOTH*	FDATA*	MIN*	RULES*	
BREAK*	Feb	MPLAN	RULVALUE*	
BROWN	FLAG*	NOECHO	RW	
CASE	FNAME*	NONE	= S*	
CHARACTER	FOOTING*	NOT	SCHEMA*	
CLEAR	FOOTINGS*	Nov	SCRATCH	
COLNAME1*	FOR*	NULL	SCREEN*	
COLNAME2*	FOREGRND	NUMRULE*	SECTION	
COLOR	FORM*	Oct	SEMI	
COLUMN*	FORMING*	OF	Sep	
COMMA	FROM*	OFF	SORTED*	
CONTAINS*	FUNCT*	ON	SUM*	
COUNT*	GLOBAL	OR*		
		OWNER*		

\*May not be used for column, table, or password names

## R:base Arguments

Table 5-4 shows a list of common R:base arguments. Table 5-5 describes the commonly used syntax arguments. Any unusual uses of the common arguments or arguments unique to a given command are explained with the command.

Table 5-4 Commonly Used Syntax Arguments

Argument*	Meaning
=	Where no space is shown following an equal sign, then no space precedes the equal sign. Where an equal sign is shown with a space, then a matching space is on the opposite side.
chrname	Specified characters which may be set by the user. Including: BLANK, DOLLAR, QUOTES, SEMI, PLUS, and COMMA.
chrpos	Character position within a variable (see MOVE).
cmdfile	A file containing R:base commands. It has the same format as <i>filespec</i> .
colname	Identifies the name of a column and may be 1-8 characters long. Instead of typing the column name, you can enter # <i>c</i> where <i>c</i> is the item number shown when the columns are listed. Or, you can use a dotted variable whose value is the column name.
collist	A list of column names or # <i>c</i> .
condition	A comparison which defines specific rows to be acted upon by a command. Syntax for conditions are shown with the WHERE syntax.
condlist	A list of up to 10 conditions that identify the rows to be referenced. Used with WHERE, IF...THEN, and WHILE...ENDWHILE.
commandname	Used with HELP and PROMPT. A command available to either of these modes. Not all commands have on-line HELP or PROMPT capability.
datatype	A valid R:base datatype: DATE, DOLLAR, INTEGER, REAL, TEXT, or TIME.
dbspec	A database designation in the form, <i>d:\path\dbname</i> .
dbname	A 1-7 character database name.
d:	A drive letter.
else-block	Commands to be executed if the IF condition is not met (see IF...THEN).
expression	An arithmetic formula or value, or concatenated terms (see ASSIGN, SET VARIABLE).
filename.ext	A 1-8 character file name plus the optional 1-3 character file extension <i>.ext</i> .
filespec	A unique DOS file specification in the form <i>d:\path\filename.ext</i> .
formname	Specifies the name of a table form and may be 1-8 characters long.
keylist	One or more special key designators used to indicate when to exit from an application. These keys are specified as ESC, ENTER, PGUP, and PGDN.
keyword	An R:base function used to determine the R:base environment. These include AUTOSKIP, BELL, CASE, CLEAR, COLOR, DATE, ECHO, ESCAPE, LINES, MESSAGE, NULL, REVERSE, RULES, SCRATCH, USER, and WIDTH. See SET and SHOW.
labelname	The name with LABEL to indicate where to skip to when a GOTO is executed.

Continued Next Page

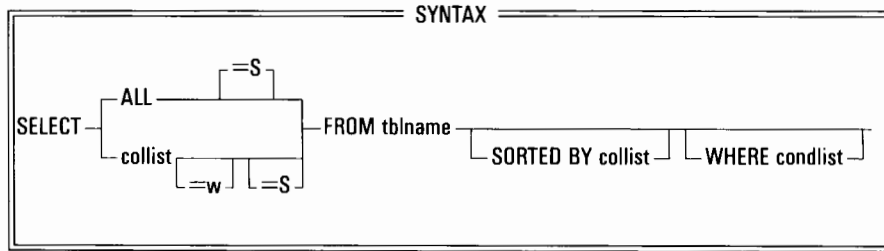
Table 5-4 Commonly Used Syntax Arguments (*continued*)

Argument*	Meaning
length	The number of characters for a TEXT datatype column or variable.
listtype	LIST types are: DATABASES, TABLES, COLUMNS, RULES, FORMS, REPORTS, tblname.
logicalop	Any of the operators: EQ, NE, LT, LE, GT, or GE.
menuname	Specifies the name of a menu and may be 1-8 characters long.
message	Message to be displayed on the screen.
newdbspec	New database specification.
#n	Specifies a route number where n is 1, 2, or 3 as determined by the SET POINTER command.
nchar	Number of characters (see MOVE).
paramlist	Parameter list. A list of up to nine values to be passed to a command file.
password	A 1-8 character OWNER, read, or write password.
path	A series of one or more directory names that lead from the root directory to a specified directory.
profile	A binary procedure file created by RCOMPILE.
r	An integer designating number of rows.
(row,column)	Multiplan row and column used with LOAD.
rptname	Specifies the name of a report and may be 1-8 characters long.
scrnname	Specifies the name of a screen and may be 1-8 characters long.
scrnrow/scrncol	The terminal screen rows and columns.
tblname	Specifies the name of a table and may be 1-8 characters long.
then-block	Commands to be executed if the IF condition is met (see IF...THEN).
value	A constant amount usually to be entered into a column or variable.
varformname	A 1-8 character variable form name.
varname	Specifies a variable name and may be 1-8 characters long.
varlist	Specifies a list of variables.
=w	An equal sign followed by an integer specifying the display width for a column or variable.
while-block	Commands to be executed if the WHILE condition is met (see WHILE...ENDWHILE).

\*If syntax uses more than one argument, each argument is numbered—*argument1*, *argument2*, and so on. For example, *tblname1*, *colname2*, *chrpos2*.

## Entering R:base Commands

The following command is an example of a typical R:base command that you would see in this manual. Throughout the manual, commands and keywords are printed in upper case, and arguments are printed in lower case. Enter the command syntax exactly as it is shown and substitute your own arguments for those shown in lower case type. You may enter command lines using either upper or lower case type unless you modify the CASE command that is explained later in this chapter. Optional portions of commands are offset from the command line.



The keyword-argument combinations vary with each application of this command. In the previous example, the table and column names are arguments whose values must be filled in.

## Command Comments

Comments may be inserted anywhere within a command line by enclosing an entry between parentheses preceded by an asterisk. For example, if you wish to insert a comment such as “end of first while loop,” at the R> prompt type:

```
*(END OF FIRST WHILE LOOP) ENDWHILE
```

Comments are useful when a series of commands are in a command file. A comment can be used to identify the purpose of a command line or the location of a routine. For additional information on command comments, see chapter 15.

## Abbreviating Commands

Commands and keywords that are comprised of four or more letters can be abbreviated to three letters when a command line is entered. For example, the abbreviation for INTERSECT is INT. If the first three letters of a command or keyword are not unique, the abbreviation must be at least four letters long. For example, the first three letters of the commands ENDIF and ENDWHILE are the same. Therefore the commands can be abbreviated to ENDI and ENDW.

## Entering Commands Longer Than 79 Characters

You can enter a maximum of 79 characters on a single line with an R:base prompt. If an entry is longer than 79 characters, two or more display lines are required. A plus (+) sign at the end of a line extends the command line to the next display line. The plus sign must be the last entry on the line.

For example, the command

```
R>SELECT ALL FROM transx SORTED BY product WHERE product EQ PX3050
```

is equivalent to:

```
R>SELECT ALL FROM transx SORTED BY product WHERE +  
R>product EQ PX3050
```

The previous example used the plus sign to extend the command line at the space between two words. If the plus sign is used without a space before it the entry is continued, but you see a +> prompt instead. For example:

```
R >SELECT ALL FROM transx SORTED BY PRO+  
+>DUCT WHERE product EQ PX3050
```

## Entering More Than One Command or Data Row on a Line

You can enter more than one command or data row on the same display line by separating each row with a semicolon (;).

For example, the commands:

```
R>LIST TABLE ALL  
R>SELECT ALL FROM tblname
```

are equivalent to:

```
R>LIST TABLE ALL;SELECT ALL FROM tblname
```

In the data entry mode, the entries:

```
L>"BOB" "BILL"
L>"SUE" "TOM"
```

are equivalent to:

```
L>"BOB" "BILL";"SUE" "TOM"
```

## Reentering the Previous Command

R:base can recall the last command or data entry made. This is useful when you must make similar entries or correct a mistake.

There are three ways to recall the previous input line:

- Simultaneously pressing the [Ctrl] and right-arrow keys recalls the entire first line.
- Pressing the right-arrow key recalls the first line one character at a time.
- Pressing the asterisk key twice recalls an entire command regardless of how many lines it occupies. Entering a single asterisk, or *\*n*—where *n* is a number—recalls portions of a command.

Use of the [Ctrl] and right-arrow keys is sufficient if only one line of input needs to be recalled. For example, if the last command line you entered was

```
SELECT ALL FROM transx
```

and you misspelled SELECT, you can correct the error in this way:

1. Press the right-arrow key four times to place the cursor on the *C*
2. Press the [Ins] key once to provide a space for the *E*
3. Press the [E] key once to insert the *E*
4. Simultaneously press the [Ctrl] and right-arrow keys
5. Press [ENTER]



If a command is longer than one line, the asterisk key can be used to recall the entire command or only portions of it.

- \* Repeats an item.
- \**n* Repeats the next *n* items (for example, \*7 repeats the next 7 items).
- \*\* Repeats the item and all remaining items in the command.

For example, if your previous entry was

```
SELECT ALL FROM transx
```

each of the commands in the following screen recall the same command:

```
R>SELECT ALL * transx
R>SELECT *2 transx
R>**
```

The asterisk function is helpful when you make an error while entering a multiple line command. For example, suppose the last command you entered was:

```
D>tblname with col1 col2 col3 col4 col5 col6 col7 +
D>col8 col9 col10 col11 col12 col133 col14 col15 col16 +
D>coll17 col18 col19 col20
```

Columns 13 and 17 were miskeyed as col133 and coll17. To correct the errors:

1. Type: \*14 (the first 14 entries were correct. The plus sign is not counted as an entry)
2. Type: col13 (correcting col133 to col13)
3. Type: \*3 (the next 3 entries were correct)
4. Type: col17 (correcting coll17 to col17)
5. Type: \*\*[ENTER] (all other entries were correct)

Your entry at the D> prompt looks like this:

```
*14 col13 *3 col17 **
```

## COMMAND CLAUSES

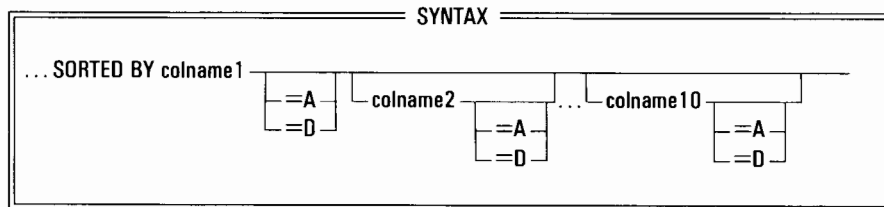
### Sorting Information with a SORTED BY Clause

You can add a SORTED BY clause to commands so that R:base organizes the information in a way that is meaningful to you. Columns can be sorted alphabetically or numerically, in ascending or descending order. If the SORTED BY and WHERE clauses are both used with a command, the SORTED BY clause must precede the WHERE clause.

Up to ten columns may be specified for a sort. If more than one column is specified, the data is sorted by the first column until two or more identical items appear. Those items are sorted by the second column specified in the row. Identical items in the second column are sorted by the third column specified, and so forth. If only one column is specified, identical items in the column are sorted in the order that they were entered. The following screen displays the result of a typical two column sort.

Sorting is accomplished in either ascending or descending order. Ascending order starts with the letter *A*, the oldest date, the earliest time, or the smallest number and ends with the letter *Z*, the most recent date, the latest time, or the largest number. Descending order is exactly the opposite. TEXT type columns that contain entries starting both with letters and other characters are sorted first by special characters, then numerically, and last alphabetically. If no sort direction is specified, R:base sorts the data in ascending order.

The syntax for the SORTED BY clause is:



You can specify ascending (*=A*) or descending (*=D*) order. Enter the appropriate letter, *A* or *D* after the equals sign.

For example, if you want to look at a transaction file named *transact* to find out what sales representatives have been selling items recently, you might want to sort the names in ascending order and to sort the transaction date in descending order. This would alphabetize the names and show the most recent sales for each person first. Your entry for the command and the R:base response look like this:

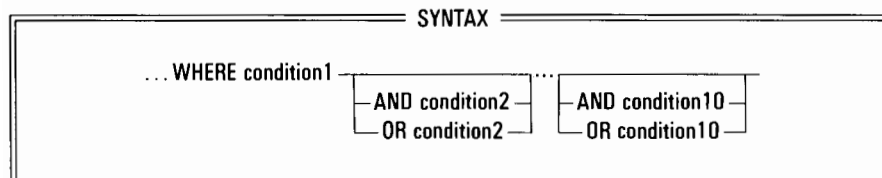
```
R> SELECT sname tdate FROM transact SORTED BY sname=A tdate=D
```

sname	tdate
Hernandez	02-15-85
Hernandez	02-05-85
Hernandez	01-25-85
Hernandez	01-15-85
Simpson	02-27-85
Simpson	02-20-85
Simpson	02-01-85
Simpson	01-29-85
Simpson	01-22-85
Simpson	01-09-85
Smith	01-30-85
Smith	01-18-85
Smith	01-07-85

### Qualifying Information with a WHERE Clause

You can add a WHERE clause to R:base commands to specify additional selection conditions. Use the WHERE clause to compare a column with a value, to compare two columns, or to locate specific information. The operating system wildcard characters can also be used with R:base. For example, you can use the global characters *?* and *\**. The *?* character can be used in place of any one character in a column name. The *\** character can be used in place of a single character and the characters that follow.

The WHERE clause is made up of the word *where* and a condition. The syntax for the WHERE command is:



The condition is similar to a simple arithmetic equation that has a mathematical operator in between two values. For example, a common condition is:

```
colname1 eq 1500
```

Table 5-5 lists the operators used to compare a column with a value (or variable), table 5-6 lists the operators used to compare two columns, and table 5-7 lists clauses used to locate specific information.

Note that when a WHERE clause is used with the JOIN command, the syntax is different from that shown in this section. For the correct syntax, see chapter 10.

Table 5-5 WHERE Clause Operators Used to Compare a Column with a Value

Operator	Symbol	Definition
EQ	=	Equal
NE	< >	Not equal
GT	>	Greater than
GE	> =	Greater than or equal to
LT	<	Less than
LE	< =	Less than or equal to
CONTAINS		Contains a text value

If you compare a column with a value, you can enter the value or you can specify a global variable instead. If you specify a variable, R:base compares the column with the most current value of the variable. Precede the variable name with a period when you enter it to indicate that you are using the value of the variable. For example, if you want to list sales amounts that are less than the value of the variable *dailyave*, type:

```
WHERE amount LT .dailyave
```



Table 5-6 WHERE Clause Operators used to Compare Two Columns

Operator	Symbol	Definition
EQA	=A	Value of colname1 equals the value of colname2
NEA	< >A	Value of colname1 does not equal the value of colname2
GTA	>A	Value of colname1 is greater than the value of colname2
GEA	>=A	Value of colname1 is greater than or equal to the value of colname2
LTA	<A	Value of colname1 is less than the value of colname2
LEA	<=A	Value of colname1 is less than or equal to the value of colname2

Table 5-7 WHERE Clauses Used to Locate a Specific Information

Operator	Definition
CONTAINS value*	Contains a text value
EXISTS	Column contains data
FAILS	Column has a null value
COUNT EQ n	Finds row <i>n</i> (an integer value) in the table
COUNT EQ LAST	Finds the last row in the table
LIMIT EQ n	Limits the number of rows that are found to <i>n</i> rows

\*You can use *NE \*string\** to form an effective NOT CONTAINS operator where \* indicates wildcard characters and *string* is the text value. CONTAINS is equivalent to *EQ \*string\**.

The LIMIT EQ condition lets you specify the number of rows to be found. It does not count as one of the ten WHERE conditions if it is specified last. For example, you can specify ten WHERE conditions and then specify the LIMIT EQ condition.

### ENTERING MORE THAN ONE WHERE CONDITION

When you use more than one WHERE condition with a command, the clauses are connected using the logical operators AND and OR. You can include up to ten comparisons in one command. The AND operator requires that the two conditions it separates be satisfied. The OR operator requires that only one of the two conditions it separates be satisfied.

As an example of the AND operator, suppose you want to search the *product* table in the *compuco* database for products with an on-hand quantity of more than 100 and a cost of \$1,000 or more. Your command looks like this:

```
SELECT ALL FROM product WHERE onhand GT 100 AND cost GE 1000
```

As an example of the OR operator, suppose you want to search the transaction table (*transx*) in the *compuco* database for customers who purchased either an Advanced PC or a Superior PC from COMPUCO. The product numbers for the two computers are *MB3020* and *MB3030*. Your command looks like this:

```
SELECT custid FROM transx WHERE prodid EQ MB3020 OR prodid EQ MB3030
```

If you have a clause that requires both operators, the order in which you list the conditions affects the information that R:base displays. Conditions are processed from left to right. Be sure to test complex clauses and verify the result. The following command line is an example of a command that uses both operators.

```
SELECT ALL FROM transx WHERE empid = 102 OR empid = 131 +  
AND units GT 25 AND tdate GT 03/11/85
```

### THE WHERE CLAUSE AND KEY PROCESSING

You can significantly reduce the time R:base takes to process a WHERE clause by using key processing. Time may be reduced if all three of the following conditions exist:

- The last condition in the clause must refer to a key column
- The last operator used in the condition must be EQ or =
- If more than one condition is specified in the clause, the last condition must be preceded by the logical operator AND

For additional information on how to define columns as key columns, see chapter 6.

### ENTERING SORTED BY AND WHERE CLAUSES FOR THE SAME COMMAND

If you use both the SORTED BY and WHERE clauses with a command, enter the SORTED BY clause before the WHERE clause. For example, if you want to list numerically all COMPUCO portable computers, your command looks like this:

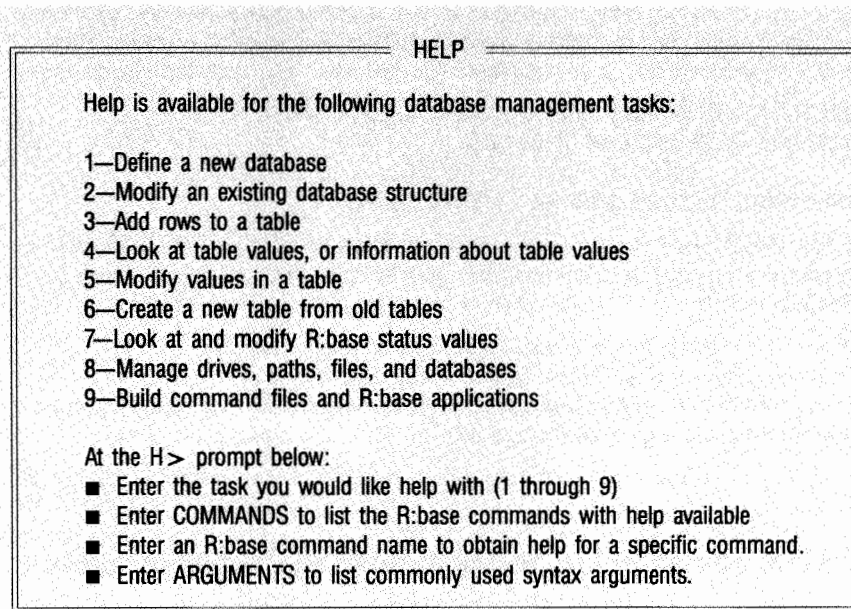
```
SELECT ALL FROM product SORTED BY prodid WHERE prodid CONTAINS p
```

## DATABASE COMMANDS

The commands listed in this section are fundamental to R:base operation. The functions that they execute apply to all aspects of operation and may be entered in any phase of R:base operation.

### Requesting On-Line Help with HELP

Press [F10] to request on-line information about R:base commands, or enter the HELP command. When you request the HELP mode, a screen similar to the one shown below appears.



For additional HELP text, enter a command name. To leave HELP, enter END.

For the previous HELP menu press [ESC]

H>

If you would like help for a specific command, at the R> prompt type the command name immediately after the word *help*. For example, if you want help with the SELECT command, at the R> prompt type:

HELP select

The screen shown below appears.

**SELECT**

Use SELECT to retrieve and display rows of data from a table.

- All column data types may have =w appended, and DOLLAR, INTEGER, or REAL data types may have =S appended.  
   =w determines the display width (w)  
   =S specifies summation of the column or columns
- Enter a list of column names to display specific columns in a table or to change the display order of the columns.

**SYNTAX**

SELECT

ALL

collist

=w

=S

FROM tblname . . .

...  
[ SORTED BY collist ]

[ WHERE condlist ]

For additional HELP text, enter a command name. To leave HELP, enter END.  
 For the previous HELP menu press [Esc].  
 H>

Some help explanations require more than one screen to display the entire help text. When a continuation screen exists, the display stops after each full screen. If you want to read the additional text, press any key other than [ESC]. If the additional help text is not needed, press the [ESC] key.



If help is requested for a command or keyword that has no help text (for example, WHAT), a response similar to the one shown below appears.

```
H>WHAT
  There is no help text for the keyword WHAT
  For additional HELP text, enter a command name. To leave HELP, enter END.
  For the previous HELP menu press [ESC].
H>
```

When you need no additional help, at the H> prompt type:

```
END
```

R:base displays an R> prompt when you exit the HELP mode.

## Requesting On-Line Prompts with PROMPT

The PROMPT command provides on — line assistance for using R:base commands. Enter the prompt mode from the R:base entrance menu or request the prompt mode for a specific command.

To list all of the commands that have PROMPT assistance available, at the R> prompt type:

```
PROMPT
```

A screen similar to the following appears.

```
---Edit menu---Table list---Column list---Go execute---Help---Quit-----
```

### PROMPTS

Prompts are available for the following commands:

APPEND	DISPLAY	INTERSECT	PASSWORDS	RULES	UNION
ASSIGN	EDIT	JOIN	PRINT	RUN	UNLOAD
BUILD	ENTER	LIST	PROJECT	SELECT	USER
CHANGE	EXIT	LOAD	RBEDIT	SET	
COLUMNS	EXPAND	OPEN	RELOAD	SHOW	
COMPUTE	FORMS	OUTPUT	REMOVE	SUBTRACT	
DEFINE	HELP	OWNER	RENAME	TABLES	
DELETE	INPUT	PACK	REPORTS	TALLY	

Press [G] to use Prompt mode.

In response to the menu displayed at the top of the screen, press [G] to enter the PROMPT mode. A P> prompt is displayed. Enter the command name for which you want prompts.

When you are in the PROMPT mode, the menu displayed at the top of the screen offers the following options:

- *Edit menu* returns you to the prompt screen last displayed, giving you the option to change your entries before the command is executed
- *Table list* displays the tables in the database
- *Column list* displays the columns in a table
- *Go execute* directs PROMPTS to process the command
- *Help* displays additional text about the command
- *Quit* returns you to an R> prompt

To request prompting for a specific command, type the command name immediately after the word *prompt*. For example, if you enter

PROMPT select

at an R> prompt, a screen similar to the following appears. After you enter the information, press [ESC] to return to the PROMPT menu.

Press [ESC] when done with this data

PROMPTS

SELECT

Use the SELECT to retrieve data from a table. Specify the table and the columns. To display all columns, enter ALL. To sum a column, add =S to the column name. To set a display width for a column, add =w to the column name, where w is a number from 1 to 256.

Name of table to display :

Name(s) of columns to display (or ALL) :

Sort order (optional list of column names) :

Condition that selects rows to display (optional) :

The amount of highlighted space displayed after the prompt varies depending upon the type of information to be entered but never exceeds 73 characters. If you need to enter more than 73 characters (for example when you define a table with several columns), you must enter the commands directly instead of using the PROMPT mode.

To exit the PROMPT mode, at the P> prompt type:

END

## Opening a Database with OPEN

The OPEN command locates a previously defined database and permits you to view its contents. You must open a database in order to review or modify the database structure or contents, or to print reports. To open a database, at the R> prompt type:

OPEN dbspec

If the database is not stored in the current directory, add the drive and subdirectory before you type the database name. Only one database may be open at a time. An open database is closed when you:

- Open another database
- Execute the CLOSE command
- Execute the EXIT command

For example, if the database *compuco* is open and you type:

OPEN c:\tools\wp\letters

*compuco* is closed and *letters* is opened.

## Closing a Database with CLOSE

The CLOSE command terminates access to the open database. Data stored in memory buffers is written to the disk and the database files are closed. Use the CLOSE command to close a database if you want to open a database stored in a different subdirectory or on a different disk.

At the R> prompt type:

CLOSE

## Exiting R:base with EXIT

The EXIT command is used to leave R:base and return to the computer's operating system. To exit R:base, at the R> prompt type:

```
EXIT
```

If you operate R:base using floppy disks, always enter the EXIT command before you remove the disk(s) from their drive(s) and turn off the computer. If you do not exit R:base in this way, the open database can be destroyed.

## Clearing the Screen with NEWPAGE

The NEWPAGE command sends a form-feed character to your current output device. If your output device is the computer screen, typing NEWPAGE clears the screen and moves the R> prompt and cursor to the top of the screen. If your current output device is a printer, NEWPAGE causes the paper to advance one page.

When you want to clear the screen or advance the paper, at the R> prompt type:

```
NEWPAGE
```

## Identifying the Database User with USER

The USER command operates with the PASSWORDS command to regulate access to the database. For information about the PASSWORD command, see chapter 6. If you have defined passwords for your database, R:base requires that users identify themselves with the USER command before password-protected operations can be performed. To enter the command, use the following syntax at the R> prompt:

```
USER password
```

## Identifying the Source of Incoming Information with INPUT

The INPUT command determines the source of incoming information. You can input data from a keyboard or an existing file. The command

```
INPUT KEYBOARD
```

informs R:base that the computer keyboard is the input source. The command

```
INPUT filespec
```

informs R:base that a file is the input source. If no drive or path is specified before the file name (for example, *d:\system\file*), R:base searches for the file on the current drive and directory.

## Identifying the Destination of Information with OUTPUT

The OUTPUT command determines the destination of information. You can direct data to a printer, a file, a computer screen, or any combination of the three.

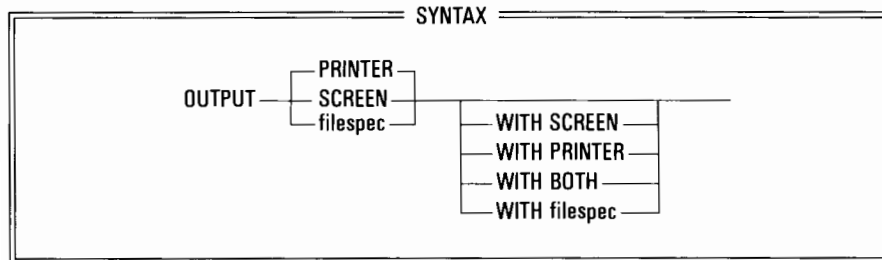
The following commands direct output to a single destination:

- **OUTPUT PRINTER** directs output to the printer.
- **OUTPUT filespec** directs output to a file. If no drive or path is specified before the file name (for example, *d:\path\file*), the file is output to the default drive and current directory.
- **OUTPUT SCREEN** directs output to the display screen. This option is the R:base default setting.

The following commands direct output to more than one destination:

- **OUTPUT filespec WITH SCREEN** directs output to a file and to the display screen.
- **OUTPUT filespec WITH PRINTER** directs output to a file and to the printer.
- **OUTPUT filespec WITH BOTH** directs output to a file to the display screen, and to the printer.
- **OUTPUT PRINTER WITH SCREEN** directs output to the printer and to the terminal screen one printer page at a time. Printing a multi-page report requires pressing any keyboard key when the printer stops at the end of each page. For additional information on printing reports, see chapter 11.

To execute the OUTPUT command, use the following syntax at the R> prompt:



If an output file is open, it is closed when the output device is changed. If the file is reopened later, R:base starts writing at the beginning of the file and erases all previously stored information.

To close an output file, use the OUTPUT command to direct information to a screen or printer. For example, at the R> prompt type:

**OUTPUT SCREEN**

You can also use the OUTPUT command with the SET ECHO command. When output is directed to a file and ECHO is set on, all R:base commands that are entered are also saved in the output file. If ECHO is set off, all output except for R:base commands is saved. For additional information on the ECHO command, see "Setting System Default Values" in this chapter.

When the **REMOVE** or **DELETE** commands are used to remove tables or delete rows, the information is no longer stored in the database. The disk space that was occupied by the deleted information, however, is unusable because R:base recognizes previously occupied space as assigned space. The **PACK** command reorganizes the database and frees the unusable disk space.

```

    SYNTAX
    PACK [ dbspec ]
  
```

You are about to perform an in-place PACK of your database.  
You should back up this database before proceeding. Continue (Y/N)

The **PACK** command is similar to the **RELOAD** command explained later in this chapter. Use **PACK** to recover unusable disk space. Use **RELOAD** to recover unusable disk space and make a duplicate copy of your database.

The SET command directs R:base to modify the default values for special characters and operating conditions, and to create or change the following: special characters, status functions, and variables. Each time R:base is started, the program resets all default values (except the date format) to the system default values. The system date and time variables (*#date* and *#time*) are set to reflect the current date and time.

**SPECIAL CHARACTERS**

R:base uses special characters to separate data fields and interpret data input. The R:base special characters are: BLANK, COMMA, DOLLAR, PLUS, SEMI (semi-colon), and QUOTES (quotation marks). The default values for these characters are set to the most common usage.

If your database requires that any of the special characters mean something other than the R:base default definitions, you must change the default value for that special character.

For example, R:base recognizes quotation marks as text-string separators. If a file that contains TEXT entries with quotation marks is loaded into R:base, the file is loaded incorrectly. Setting QUOTES to something other than quotation marks, such as an exclamation point (!), solves the problem. Note that you must also edit the input file and insert exclamation points as string separators before you attempt to load the file. To instruct R:base to interpret an exclamation point as a text string separator, at the R> prompt type:

```
SET QUOTES=!
```

The following screen displays the result when three rows of data are loaded.

```
L >!Based on the article "Hawaiian Connection"! !The Seattle Bugle! !p A10!  
L >!Direct quote from Sam Evans, President of COMPUCO! !N/A! !N/A!  
L >!Interpreted from the paper "The Expanding Marketplace"! !COMPUCO System+  
+>Journal! !p 219!
```

**Setting Special Characters**

The syntax for entering special characters is:

```
SET chrname=value
```

Note that there are no blank spaces on either side of the equals sign. For example, if you want to change the value for quotes to a circumflex (^), at the R> prompt type:

```
SET QUOTES=^
```

Do not set the value for any of the special characters equal to the default value of another special character. If two characters have the same value, R:base uses the characters incorrectly because they are reserved for command syntax interpretation.



## OPERATING CONDITIONS

The R:base operating conditions that can be set are related to the keywords: autoskip, bell, clear, color, date format, echo, error messages, escape, lines per page, messages, null value indicator, reverse video, rule checking, scratch files, upper/lower case distinction, and width of the screen.

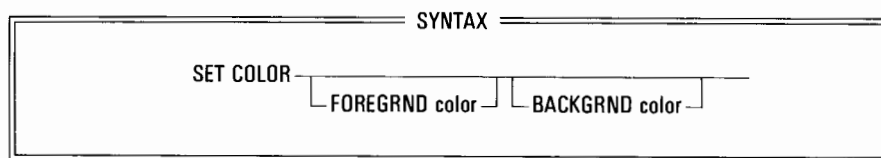
### Setting Keyword Values

**Autoskip.** When you are entering data with a table form, SET AUTOSKIP ON directs R:base to move automatically to the next column as soon as all spaces in the current column are filled. For instance, if a column is defined as eight characters long, and your entry is only six characters, you must enter the last two characters as blanks in order for AUTOSKIP to function. SET AUTOSKIP OFF requires pressing [ENTER] after each entry. The R:base default value is OFF.

**Bell.** SET BELL ON causes the terminal's speaker to sound when an error is made. SET BELL OFF silences the speaker. The R:base default value is ON.

**Clear.** SET CLEAR ON directs R:base to clear the internal buffers and transfer the database files to the disk after each data modification command is completed. SET CLEAR OFF directs R:base to store modified database records in a buffer and write modified data to the disk when the buffer is full or when the R:base mode is exited. If the CLEAR function is off, repetitive modifications to a database may run faster, but all of the changes in the buffer will be lost if an accident, such as a power supply fluctuation, occurs. The R:base default value is ON.

**Color.** You may enter the SET COLOR command if your computer has a color graphics board. The syntax for the SET COLOR command is:



**FOREGRND colors:** Black, Blue, Brown, Cyan, Gray, Green, Magenta, Red, Light Black, Light Blue, Light Cyan, Light Green, Light Magenta, Light Red, White, and Yellow.

**BACKGRND colors:** Black, Blue, Brown, Cyan, Gray, Green, Magenta, and Red.

When you enter SET COLOR without specifying foreground and background colors, the colors are displayed. Choose a foreground color by placing the cursor beside the color you want and pressing [C]. Use the same procedure to choose a background color. After you choose both colors, press [ESC]. Then press [S] to save your choices.



**Date.** The DATE function can be set to any of the following basic formats:

Day-Month-Year:	DD/MM/YY	DD/MM/YYYY	DD/MMM/YY	DD/MMM/YYYY
Day-Year-Month:	DD/YY/MM	DD/YYYY/MM	DD/YY/MMM	DD/YYYY/MMM
Month-Day-Year:	MM/DD/YY	MM/DD/YYYY	MMM/DD/YY	MMM/DD/YYYY
Month-Year-Day:	MM/YY/DD	MM/YYYY/DD	MMM/YY/DD	MMM/YYYY/DD
Year-Month-Day:	YY/MM/DD	YYYY/MM/DD	YY/MMM/DD	YYYY/MMM/DD
Year-Day-Month:	YY/DD/MM	YYYY/DD/MM	YY/DD/MMM	YYYY/DD/MMM

The day (DD) must be expressed as two digits. The month (MM) can be expressed as two digits (01) or as a three letter abbreviation (JAN). The year can be expressed as a two digit abbreviation (85) or as four digits (1985). The R:base default format is MM/DD/YY. R:base can accept dates from the year 1900 through 2155 AD. The two digit years are assumed to be in this century. If you want to use dates for other centuries, be sure to use a date format with a four digit year.

The delimiters (characters that separate the three parts) do not need to be entered when you set the format. Do not, however, separate the parts with blank spaces or commas. When you are loading data, the delimiters that R:base accepts are / and -. Note that you do not have to use the same delimiter each time a date is entered.

For example, SET DATE MMM-DD-YYYY instructs R:base to interpret the first three digits as month, the next two digits as day, and the final four digits as year. The command can also be entered in one of the following ways:

```
SET DATE MMM/DD/YYYY
SET DATE MMMDDYYYY
```

After the date format is set, R:base only recognizes dates entered in accordance with that format. For example, if the format is MMMDDYYYY, you can enter any of the following for June 28, 1985:

JUN 28 1985	6-28-85
JUN 28, 1985	6/28/1985
JUN/28/1985	06-28-1985
JUN 28, 85	

If a month or day is only one digit, it is not necessary to enter a zero before the digit. For example, January 1, 1985 can be entered as:

```
1-1-85
```

**Echo.** SET ECHO ON causes all commands and data from the current input device to be listed on the current output device. This is useful when output is directed to a printer, because the R:base commands are printed immediately ahead of the requested output. (Without the echo, the output would be printed without the R:base command.) SET ECHO OFF cancels the echo mode. The R:base default value is OFF.

**Error Messages.** SET ERROR MESSAGE ON directs R:base to display an error message when a system error occurs. SET ERROR MESSAGE OFF directs R:base not to display an error message when a system error occurs. The R:base default value is ON.

**Escape.** SET ESCAPE OFF lets you turn off the ability to escape or abort processing in the middle of command files, WHILE loops, and database access. SET ESCAPE ON enables you to escape. The R:base default value is ON. The ESCAPE function is used by programmers to control programming interruptions. For additional information on the command, see chapter 15.

**Lines.** The LINES function controls how many lines of data per page are directed to the output device. (An 8 1/2 by 11 inch page has 66 lines, and a typical terminal screen displays 25 lines.) The R:base default value is 20. For example:

```
SET LINES 25
```

directs R:base to output 25 lines of data to the output device. This function does not affect report generation because each report allows you to define the number of lines per page.

**Messages.** SET MESSAGES ON directs R:base to display all system messages, such as "Successful PROJECT operation 3 row(s) generated" or "End R:base Data Loading." SET MESSAGES OFF directs R:base to display only error messages. The R:base default value is ON.

**Null Value Indication.** SET NULL adjusts the R:base null value indicator to any value that consists of one to four characters. The R:base default value is -0-. To change the representation to a new value that is four or less characters, use the following syntax at the R> prompt:

```
SET NULL value
```

If you set the null value to a blank space (SET NULL " "), rows that are composed entirely of null values are not displayed by the SELECT command.

**Reverse.** The REVERSE function can be set on or off. SET REVERSE ON directs R:base to display reverse video for column entry areas on forms and prompt screens. SET REVERSE OFF directs R:base to cancel the reverse video display. The R:base default value is ON.

**Rule Checking.** SET RULES ON directs R:base to check input against all existing rules. SET RULES OFF directs R:base to ignore all existing rules. The R:base default value is ON. If you have defined an owner password for your database, R:base does not accept the SET RULES OFF command until you enter the password. For more information on rules, see chapter 6.

**Scratch Files.** SET SCRATCH ON directs R:base to store scratch files on the same drive and subdirectory as the database. SET SCRATCH OFF directs R:base to store temporary scratch files that are created when data is sorted on the default drive and subdirectory. The R:base default value is ON.

**Upper/Lower Case Distinction.** SET CASE ON directs R:base to distinguish between upper and lower case when a comparison is used with SORT, WHERE, IF, and WHILE expressions, and with the TALLY command. For instance, an upper case *A* would be interpreted differently than a lower case *a*. SET CASE OFF directs R:base to interpret upper and lower case characters as identical. The R:base default value is OFF.

**Width.** The WIDTH function controls the width of data directed to the printer when you use the SELECT command. (An 8 ½ by 11 inch page and a typical terminal screen both display 80 columns.) The width must be set between 40 and 256. Do not set the width to a number greater than the number of characters your printer can fit on one line. The R:base default value is 79. For example:

```
SET WIDTH 132
```

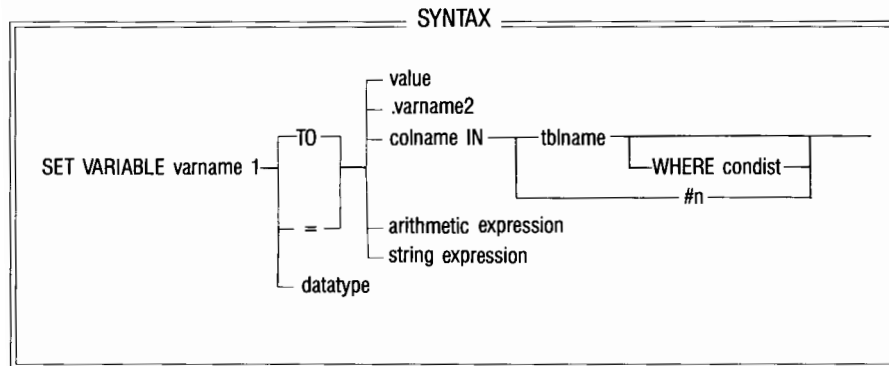
directs R:base to allow displays of up to 132 characters per line when you use the SELECT command with a wide carriage printer. This function does not affect report generation because each report allows you to define the page width.

## VARIABLES

There are three kinds of R:base variables: Global, Error, and Report. Global variables are temporary variables that exist within R:base but are not part of any database. Error variables are associated with error message numbers designed for a particular program. Report variables are used for temporary operations in the REPORTS mode. Further information about variables is also in chapters 11 and 15.

### Global Variables

The SET VARIABLES command defines or redefines a global variable name, type, and value or expression. The syntax for the command is:

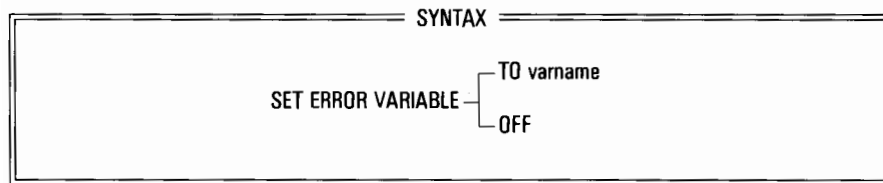


A complete explanation of the SET VARIABLE command is in chapter 15 and a summary of the command is in the *R:base 5000 Reference Manual*.

### Error Variables

The SET ERROR VARIABLE command allows programmers to write an error handling routine that traps errors.

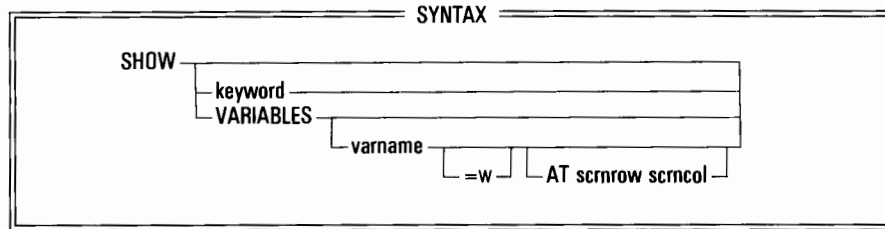
The syntax to set an error variable is:



A complete explanation of the SET ERROR VARIABLE command is in chapter 15. A summary of the command and a list of error messages is in the *R:base 5000 Reference Manual*.

## Displaying System Default Conditions and Variables with SHOW

You can use the SHOW command to display special characters, keywords, and variables defined with the SET command. The SHOW command also displays rules and the current user. To enter the SHOW command, use the following syntax at the R> prompt:



*Keywords:* autoskip, bell, clear, color, date format, echo, messages, escape, lines per page, null value indicator, reverse video, rule checking, scratch files, upper/lower case distinction, and width of the screen.

If you want to display the status of a keyword, type the keyword after the word SHOW. For example, to show the status of the keyword *autoskip*, type:

```
SHOW autoskip
```

To display all system default conditions for a database (except rules, variables, error messages, scratch files, clear, color, and current user), at the R> prompt type:

```
SHOW
```

A screen similar to the following appears:

```
Current special characters
BLANK=      COMMA=,
DOLLAR=$    SEMI=;
QUOTES="    PLUS=+

DATE Format = MM/DD/YY
LINES per page = 20
WIDTH per line = 79
ECHO of input = OFF
UPPER/lower CASE distinction = OFF
RULE checking = ON
NULL value indicator = -0-
BELL on error = ON
AUTOSKIP on data entry/edit = OFF
REVERSE video on data entry/edit = ON
Diagnostic MESSAGES = ON
```

If you want to display all the variables and their values, at the R> prompt type:

```
SHOW VARIABLES
```

If you want to display the value of a specific variable, enter the variable name after typing SHOW VARIABLES. For example, to display the value of the variable *totsales*, your entry looks like this:

```
SHOW VARIABLES totalsales
```

You can also specify the width for the display, as shown in the syntax, by adding *=w* where *w* is the maximum width of the display.

## DATABASE BACK UP AND RECOVERY

Database back up is important for ensuring you do not lose the information in your database. If the master copy of your database is destroyed or critically damaged, either because of human error or technical malfunction, a back-up copy helps you recover lost data.

Once you have created a database, make two duplicate copies and store them in different places. For example, if the original copy of the database is on a hard disk, store one back-up in a different subdirectory on the hard disk and another on a floppy disk, or store the back-ups on separate floppy disks. If the original is on a floppy disk, store the back-up copies either on separate subdirectories of a hard disk or on separate floppy disks.

Keep your back-up copies up to date. If the data is changed or updated daily, make a new back-up copy at least once a day. To keep all three copies current, make one disk the master and alternate between the other disks to make back-up copies. For example, suppose you have three disks, numbered 1, 2, and 3, that contain identical information. Make disk 1 the master copy and alternate between disks 2 and 3 to make back-up copies. At the end of one day, make a back-up copy of disk 1 on disk 2, and at the end of the following day, make a back-up copy of disk 1 on disk 3.

If you rarely change the information in the database (perhaps you make a handful of entries every week), use a rotation similar to the daily procedure previously explained to make back-up copies on a weekly basis.

You can create your back-up by using the R:base RELOAD, UNLOAD, or COPY commands, or the operating system BACKUP command. For recovery, use either the R:base INPUT or COPY commands or the operating system RESTORE command. If you use the COPY command, be sure to copy all three R:base database files. The INPUT command is explained in chapter 12. For information about the operating system commands, see your operating system manual.

## **The UNLOAD Command**

The UNLOAD command copies data from an R:base file to an ASCII file. UNLOAD can be used to copy an entire database, a single table, or a portion of a row from a table. This section pertains to database back up, and therefore only covers unloading an entire database. For further information about the UNLOAD command, see chapter 12.

The following steps outline the procedure for unloading a database to a single disk.

1. Use the OPEN command to open the database you want to back up.
2. Open an output file and assign it a name. If you want to store the file somewhere other than on the current drive and path, include the drive and path specification before the file name. For example, if you want to open a file named *store* on the *rbase* subdirectory of the *c:* drive, at the R> prompt type:

```
OUTPUT c:\rbase\store
```

3. Unload the database structure, or schema, and all the rows of data. At the R> prompt type:

```
UNLOAD ALL
```

4. Close the output file by redirecting the output to the screen. If you want to continue using R:base, at the R> prompt type:

OUTPUT SCREEN

If your database is too large to fit on one disk, you need to unload it in parts, as described below.

1. Use the DIRECTORY command to find out how much space your database needs.
2. Use the CHKDSK command to find out how much space each disk has.
3. Divide the database into logical parts, making sure that each part will fit on a disk. For example, if your database contains names and addresses, you may find that the names beginning with *A* through *G* fit on one disk.
4. Use the OPEN command to open the database you want to back up.
5. Open an output file and assign it a name. Include the drive and path specification before the file name. For example, if you want to open a file named *store* on the *b:* drive, at the R> prompt type:

OUTPUT b:store

6. Unload the database structure, or schema. At the R> prompt type:

UNLOAD SCHEMA

7. Open another output file and assign it a name. Be sure to use the same path that you specified for the output file in step 1. For example, if you want to name the file *storeA-G*, at the R> prompt type:

OUTPUT b:storeA-G

8. Unload the information in the database. In this example, the table is named *customer* and the last names are stored in the column *lastname*. At the R> prompt type:

UNLOAD DATA FOR customer WHERE lastname GE a AND lastname LT h

9. Open as many output files as you need and use the UNLOAD command to copy batches of names. When the entire database is unloaded, close the last output file you opened by redirecting the output to a printer, the screen, or another file. If you want to continue using R:base, at the R> prompt type:

OUTPUT SCREEN



## The RELOAD Command

The RELOAD command backs up a database and recovers unusable disk space. When rows are deleted, or when tables are removed, the disk space formerly occupied by the deleted information is unusable. The RELOAD command recovers unusable space by partitioning new space on a disk and compressing the empty sectors as the database is copied.

To back up an open database with the RELOAD command, at the R> prompt type:

```
RELOAD newdbspec
```

If the database is not on the current directory, include the drive and path specification before the database name. For example, *c:\rbase\dbname*.

If you wish to reload a database on the same disk as your original copy, you must assign a different name to the database when you enter the RELOAD command. For example, if the database *oldbase* is stored on a disk in the *rbase* subdirectory of drive *c:* and you want to have the duplicate copy—named *newbase*—on the same disk, at the R> prompt type:

```
RELOAD c:\rbase\newbase
```

When the reload is complete, disk *c:* will contain the databases *oldbase* and *newbase*.

The RELOAD command is similar to the PACK command that is explained earlier in this chapter. Use RELOAD to back up your database and use PACK to recover unusable disk space.

## OPERATING SYSTEM COMMANDS

R:base uses the commands similar to those commands that you normally use with your operating system. The commands are operational within R:base and although some minor differences exist, you can use them in R:base to execute similar functions. See your operating system manual for explanations of error messages.

The *R:base 5000 Reference Manual* "Command Dictionary" explains how to use these R:base commands:

CHDIR or CD	—	Change Directory
CHDRV	—	Check Drive
CHKDSK	—	Check Disk and Memory Space
COPY	—	Copy Files
DIR	—	List Directory
ERASE	—	Erase Files
MKDIR or MD	—	Make Directory
RENAME or REN	—	Rename File
RMDIR	—	Remove Directory
TYPE	—	Type ASCII Files

The operating system wildcard characters can also be used with R:base. For example, you can use the global file name characters ? and \*. The ? character can be used in place of any one character in a file name. The \* character can be used in place of a single character and the characters that follow. Note that wildcards cannot be used with the COPY command to append several files together.

## DATABASE FILES

### Contents

An R:base database consists of three operating system files. Each file has an identification number—1, 2, or 3—and extension—.RBS. For instance, the sample database named *compuco* (on your R:base 5000 disks) is made up of:

COMPUCO1.RBS  
COMPUCO2.RBS  
COMPUCO3.RBS

The first file contains the database structure definition (data dictionary) and the location of the data. The second file contains data for the database, and the third contains the index for key columns. These files operate together and should never be separated. All three files are non-text files and cannot be used directly by another program, editor, or word processor without damaging the database.

## Estimating File Sizes

You may want to estimate the size of a database to find out if there is enough storage room on the disk. The first file always consists of 9600 bytes. The size of the second and third files varies depending upon the amount of information in the database. The third file size is dependent upon the number of key columns and the number of values in those columns. As a rule of thumb, if you have one key column with 1000 values, estimate 20 bytes per row. Depending upon the structure of the tables with key columns, the estimate can be as low as 10 or as high as 37 bytes per row.

The following procedure explains how to estimate the size of the second R:base database file.

1. Calculate the row length (in bytes) for each table by adding up the length of columns for a table. Add six bytes to each row to allow for pointers. The length of columns is shown in table 5-9.

Table 5-9 Column Lengths Used to Estimate File Sizes

Column Type	Size (bytes)*
DATE	4
DOLLAR	8
INTEGER	4
REAL	4
TEXT	number of characters†
TIME	4

\*All storage is in two byte segments. Odd segment lengths are rounded up to an even number.

† The minimum text length stored is 4 bytes.

2. Divide 1536 by the number of bytes in a row for each table to find the number of rows per block. Round the result up to the next highest integer.
3. Estimate the number of rows for each table.
4. Divide the estimated number of rows by the results of step 2 for each table to find the number of blocks per table. Round the result up to the next highest integer.
5. Multiply the total number of blocks by 1536. The result is the number of bytes in file 2.

## Estimating the Size of a Sort File

R:base creates temporary scratch files for sorting large amounts of data. Temporary files are deleted after the sort is completed, but there must be sufficient room on the disk when the sort is performed. Temporary files are denoted by the extension `$.$$$`.

To estimate the amount of disk space you will need for a sort file:

1. Use table 5-9 to determine the length of the columns being sorted. Add 4 to the length to determine the total column length.
2. Multiply the number of rows being sorted times the result in step 1.
3. Multiply the result in step 2 times 2.



## Database Structure Contents



<b>How to Use This Chapter</b>	6-2
<b>Database Definition</b>	6-2
Options for Defining a Database	6-2
The Application EXPRESS	6-4
R:base Define Mode	6-5
<b>Defining the Database Structure</b>	6-5
Step 1: Defining Database Names	6-5
Step 2: Defining Column Names and Data Types	6-6
Step 3: Defining Tables	6-8
Step 4: Assigning Passwords to a Database	6-9
Owner Passwords	6-9
Table Passwords	6-10
Step 5: Defining Data Entry Rules	6-11
Leaving the DEFINE Mode	6-15
<b>Redefining the Database Structure</b>	6-15
Changing Column Definitions with CHANGE COLUMN	6-16
Adding Columns to a Table with EXPAND	6-17
Creating and Deleting Column Keys with BUILD/DELETE KEY	6-17
Removing Columns from a Table with REMOVE COLUMN	6-18
Removing Tables from a Database with REMOVE TABLE	6-18
Renaming Columns, Passwords, and Tables with RENAME	6-19
Deleting Rules with DELETE	6-20
<b>Displaying the Database Structure</b>	6-20
Displaying Database Names with LIST DATABASES	6-21
Displaying Table Names with LIST TABLES	6-21
Displaying Table Structure with LIST	6-22
Displaying Columns with LIST COLUMNS	6-24
Displaying Rules with LIST RULES	6-24
Displaying Form Names with LIST FORMS	6-25
Displaying Report Names with LIST REPORTS	6-25

## **HOW TO USE THIS CHAPTER**

In R:base, a database is a collection of tables containing data arranged in columns and rows. This chapter describes how to define and modify the tables and columns that make up the database structure, how to protect the database structure and contents with rules and passwords, and how to display the database structure on the screen. Chapter 7 explains how to enter data into the tables and columns you have defined. If the subject of database design is new to you, be sure to read chapter 2 before reading this chapter.

## **DATABASE DEFINITION**

### **Options for Defining a Database**

You can define a database with the Application EXPRESS or with the R:base DEFINE mode. Table 6-1 shows the definition tasks that you can accomplish with each method.

Table 6-1 Database Definition and Redefinition:  
The EXPRESS Compared with DEFINE and R:base Commands

Task	The EXPRESS	DEFINE Mode	R:base Command
Name a database	XX		DEFINE
Define a new table	XX	TABLES	
Define a new column	XX	COLUMNS	
Define an owner password		OWNER	
Define a table password		PASSWORDS	
Define data entry rules		RULES	
Redefine a column data type for a column used in one table	XX		CHANGE COLUMN
Add a column to a table	XX		EXPAND
Rename a column	XX		RENAME COLUMN
Rename a table	XX		RENAME TABLE
Remove a column from a table	XX		REMOVE COLUMN
Remove a table	XX		REMOVE TABLE
Rename the owner password		OWNER	RENAME OWNER
Build a key for a new column		COLUMNS	
Build a key for an existing column			BUILD KEY
Delete a key for a column			DELETE KEY



Each definition method has advantages. The EXPRESS walks you through the basic steps of the define process. If you are not familiar with R:base, the EXPRESS is the data definition mode you should use.

The R:base DEFINE mode provides no on-line assistance unless you request help with the HELP or PROMPT commands. The DEFINE mode is designed for the experienced R:base user who wants to define a database quickly.

Regardless of which definition process you choose, the only limit to the number of databases that you can create with R:base is the available disk space. The structure for databases is the same for both definition methods, and databases are stored separately on your disk.

### THE APPLICATION EXPRESS

The EXPRESS enables you to define a new database or to change the definition of an existing database. To use the EXPRESS, choose *EXPRESS* from the R:base 5000 Main Menu. A screen similar to the following appears:

```

                                Application EXPRESS
                                Copyright (c) 1985 by Microrim, Inc. (Ver. ### PC-DOS)

                                Select option--[F10] for help
(1) Define a new database
(2) Change an existing database definition
(3) Define a new application
(4) Change an existing application
(5) Display file directory
(6) Exit
```

Use the EXPRESS to accomplish all database definition tasks except:

- Defining data entry rules
- Defining passwords
- Building keys

These three tasks can be accomplished using the DEFINE mode. As shown in table 6-1, the EXPRESS can also be used for many database redefinition tasks. For a complete description of the EXPRESS capabilities, see chapter 14.

## R:BASE DEFINE MODE

There are three steps you must follow to define a database with the DEFINE mode.

1. Open a file for the database and assign it a name
2. List the columns that will be in the tables
3. Group the columns into tables

After these steps are finished, you may assign optional passwords and rules. The following sections explain each step in detail.

## DEFINING THE DATABASE STRUCTURE

### Step 1: Defining Database Names

A database name can be no longer than seven characters. The name cannot begin with a number or special character and it cannot contain blank spaces. Drive and subdirectory specifications are not included in the seven letter limit. To define a database, use the following syntax at the R> prompt:

```
DEFINE dbname
```

If you want to define the database in a directory other than the current directory, include the drive and subdirectory specification before the database name. For example, if you want to open a database called *compuco* in the *company* subdirectory of your *c:* drive, your entry and the R:base response are shown in the following screen. The D> prompt indicates that R:base is in the DEFINE mode.

```
R> DEFINE c:\company\compuco
      Begin R:base Database Definition
D>
```

## Step 2: Defining Column Names and Data Types

A column is the name of a data item. Be sure to assign your columns meaningful names so that you can quickly identify the kind of information that is stored there. For example, a column named *zip* easily identifies a column for zip codes.

The COLUMNS command starts the column definition process. You name each column and define the data types, lengths, and whether or not the column is a keyed column.

The *column name*, which can be up to eight characters in length, identifies the column. Names cannot begin with numbers or special characters and cannot contain blank spaces.

The *column type* defines the kind of data that will be stored. R:base supports six data types: DATE, DOLLAR, INTEGER, REAL, TEXT, and TIME. Table 6-2 describes each type.

Table 6-2 Column Data Types

Data Type	Description
DATE	Represents the month, day, and year format established with the SET command. See chapter 5 for format information
DOLLAR	Represents dollar amounts in the range of $\pm \$99,999,999,999,999.99$
INTEGER	Represents whole numbers in the range of $\pm 999,999,999$
REAL*	Represents real numbers in the range of $\pm 9 \times 10^{+37}$ . Real numbers have a decimal value (for example, 1.414)
TEXT	Represents alphanumeric data. The default length is 8, and the maximum length is 1500
TIME	Represents the time in hours, minutes, and seconds. The format is HH:MM:SS

\*REAL numbers are stored in a binary form. Therefore the displayed value may not be the actual stored value. Use caution when using the apparent displayed REAL value in a comparison.

The *column length* is a required entry for TEXT data types and an optional entry for INTEGER and REAL data types. When entering a length for TEXT data types, enter the amount of space you need to display every character — including blank spaces — in the column. The maximum column length is 1500 characters. If no length is specified, R:base uses the default value of 8.

Do not enter column lengths for INTEGER and REAL data types unless you are using Microrim's Program Interface (PI) software. If a length is specified for a column with INTEGER or REAL data types, R:base assumes the column is an array.

The *column key* is an optional entry for all data types. If a column is identified as a key, R:base can locate rows in the column more quickly during relational operations and database searches performed with an *equal* condition in the WHERE clause. If you want to identify an existing column as a key, use the BUILD KEY command explained later in this chapter.

Columns that contain frequently searched for information should be defined as keys. The information should also be unique in most of the rows. For example, typical key columns are part numbers, employee numbers, and social security numbers. One column can contain thousands of numbers, but each of the numbers are different. An example of a column that should not be a key is marital status. All of your entries will either be *married* or *unmarried*. If a column similar to marital status is defined as a key, information searches for data in the column may actually take longer. See chapter 5 for additional information on WHERE clauses. Note too that an R:base keyed search is sensitive to upper and lower case, and, therefore, will work best when similar text data is entered in the same case.

To enter columns, at the D> prompt type:

COLUMNS

R:base responds with a D> prompt. Enter the column name, data type, length (optional), and key (optional). Use the following format:

SYNTAX	
colname datatype	<div style="display: inline-block; border-bottom: 1px solid black; width: 100px; margin-bottom: 5px;"></div> <div style="display: inline-block; border: 1px solid black; padding: 2px 5px; margin-bottom: 5px;">length</div> <div style="display: inline-block; border: 1px solid black; padding: 2px 5px; margin-bottom: 5px;">KEY</div>

Enter all of the columns in the database.

For example, if you define columns for a transaction table, you need columns for: transaction number, identification of the employee who made the transaction, identification of the customer, product that was sold, number of units sold, sale price, and transaction date. Your entry to define the columns looks like this:

```
D>COLUMNS
D>transid INTEGER KEY
D>empid INTEGER
D>custid INTEGER
D>prodid TEXT 6
D>units INTEGER
D>price DOLLAR
D>tdate DATE
```

### Step 3: Defining Tables

After you have defined all of the columns for the database, you are ready to define tables. You can add columns to tables at any time, but it is best to plan your database as thoroughly as possible and define all of the columns at one time. If you exit the DEFINE mode before tables are defined, the columns that you defined will not be saved.

A table is a collection of columns that have common characteristics. In the *compuco* database, for example, the transaction table *transx* consists of the columns *transid*, *empid*, *custid*, *prodid*, *units*, *price*, and *tdate*. All the columns contain information that logically pertains to a sales transaction.

The TABLES command starts the table definition process. You enter the table name and list the columns that it contains. A column must already be defined in order to specify it as part of a table.

When you define tables, the order in which you enter the columns is the order that R:base follows for loading data. For example, if your entry to define the transaction table looks like this:

```
transx WITH transid empid custid prodid units price tdate
```

The order of data entry would be: transaction ID number, employee ID number, customer ID number, product ID number, number of units sold, selling price, and transaction date. If you want to change the order of columns after the table is defined, use the PROJECT command that is explained in chapter 10 to make a new table.

One table can have a maximum of 400 columns and each database can have a maximum of 40 tables. Each database also can have a maximum of 400 columns, so if you define a table with 400 columns, the database that contains the table will not have room for any other tables. Note that R:base calculates the number of columns in the database by counting the number of columns in each table. For example, if you define a database with four identical tables, and each table is made up of 100 columns, your database contains 400 columns.

When you are designing your database, remember that *forms*, *reports*, and *rules* are separate tables. If these three tables are in your database, you can have a maximum of 37 other tables in the database.

To enter tables, at the D> prompt type:

**TABLES**

R:base responds with a D> prompt. Use the following format to enter tables:

tblname WITH collist

Enter all of the tables in the database.

For example, if you define the transaction table that is discussed in this section, your entry looks like this:

```
D> TABLES
D> transx WITH transid empid custid prodid units price tdate
```

## Step 4: Assigning Passwords to a Database (Optional)

You can define two kinds of passwords that restrict access to the database: owner and table. The owner password controls access to the database structure. If you define an owner password, only persons with that password can use the DEFINE, RELOAD, and UNLOAD commands. The table password restricts access to tables. A table can have no password, a modify password (MPW), a read password (RPW), or both an MPW and an RPW. If your database requires table passwords, you first must define an owner password.

After passwords have been defined, operators must identify themselves with the USER command when they start up R:base. If you are the database owner and you are in the DEFINE mode (a D> prompt is displayed), you can also enter your password with the OWNER command. The format for the OWNER command is exactly the same as the format for the USER command. See chapter 5 for a complete explanation of the USER command.

### OWNER PASSWORDS

The OWNER command enables you to define a code number or word (up to eight characters in length) that restricts access to the database structure. The code cannot contain blank spaces. If you specify an owner password, operators must enter the password to modify the database structure with the DEFINE, RELOAD, and UNLOAD commands. Operators without the code are only able to view or modify the data in the database.

Be sure to keep a record of the owner password in a safe place away from your computer. If you lose the owner password, it is impossible to search the database structure to determine the password. You also cannot change the database structure.

To specify an owner password, use the following syntax:

**OWNER password**

For example, if you want to define *xbillxxx* as an owner password, at the D> prompt type:

**OWNER xbillxxx**

You can change an owner password at any time using the R:base **RENAME OWNER** command. Use the following syntax at the R> prompt:

**RENAME OWNER password1 TO password2**

If you have specified an owner password and later decide that the password is not needed, it is possible to remove the password entirely. Use the following syntax at the R> prompt:

**RENAME OWNER password1 TO NONE**

Additional information on the **RENAME** command is in the “Redefining the Database Structure” section of this chapter.

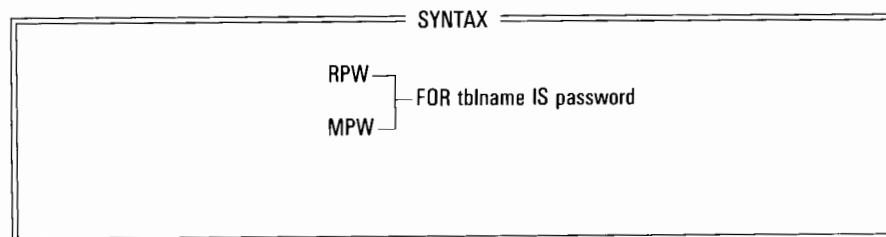
### **TABLE PASSWORDS**

Table passwords restrict access to the information in a specific table. You can define two different types of table passwords: modify (MPW) and read (RPW). An MPW allows a user to read and modify the contents of a table. An RPW only allows a user to read the contents of a table. You can define one MPW and one RPW to each table. If you want to define an RPW, you must first define an MPW.

To define table passwords, at the D> prompt type:

**PASSWORDS**

R:base responds with a D> prompt. Choose the type of password you wish to define (RPW or MPW) and use the following format:



For example, if you define *zzjackzz* as a modify password for the *transx* table, your entry looks like this:

```
D> PASSWORDS
D> MPW FOR transx IS zzjackzz
```

If you want to assign a new MPW or RPW for a table, follow the previous procedure as if you were defining a the password(s) for the first time. If you want to remove an MPW or RPW, follow the definition procedure and specify none as the password.

### Step 5: Defining Data Entry Rules (Optional)

The RULES command enables you to specify conditions to which the data must conform. The rules are checked during all data entry and modification procedures (CHANGE, EDIT, ASSIGN, ENTER, and LOAD). If the conditions of the rule are not met, an error message that you have specifically defined for that rule is displayed.

You define error messages when you define rule conditions. The message can be from one to 40 characters long and must be enclosed in quotation marks. The first word in the error message cannot be an R:base command or keyword. A complete list of commands and keywords is included in chapter 5.

One rule can apply to every table in a database or only to certain tables. When you define a rule, you specify which table(s) it applies to. You can specify up to 20 rules for one table.

Rules can be defined for many different types of situations. You can use rules to:

- Prevent duplicate information from being entered. For example, a new stock number cannot be the same as an existing one.
- Define a value range. For example, an employee number must be between 1000 and 15000.
- Verify that the data being entered corresponds with data elsewhere in the database. For example, the product stock number must exist in an on hand table in order to be entered in a sales transaction table.



You can specify up to ten conditions for one rule. Additional conditions are added using the operators AND and OR. AND requires that both conditions must be met. OR requires that at least one of the conditions must be met. If more than one condition is specified, confirmation of each rule is done in the order that they are defined. If you need to define a rule that uses both operators, enter all the conditions that use one operator. Then enter all the conditions that use the other operator.

If more than ten conditions are required, you need to define a special verification table. For example, to check if a column named *state* has a valid state abbreviation, you can define a table that consists of one column (named *abbrev*). List the abbreviations for all 50 states in *abbrev* and define a rule that requires entries in *state* to equal one of the rows in the *abbrev* table.

When you specify a rule, you list one or more conditions to which the items in a column must conform. If the items do not conform to the conditions, the error message that you defined is displayed. For example, suppose you have a table for sales transactions (*transx*) and you wish to prevent a zero entry in the column that records the number of units sold (*units*). You need to define a rule that displays an error message if the number of units entered equals zero. The condition to use in this case is *not equal*, because you want an entry to not equal zero. Your command for the rule looks like this:

"Invalid entry for number of units" units IN transx NE 0

Table 6-3 shows the conditions you can specify to compare a column with any value other than another column.

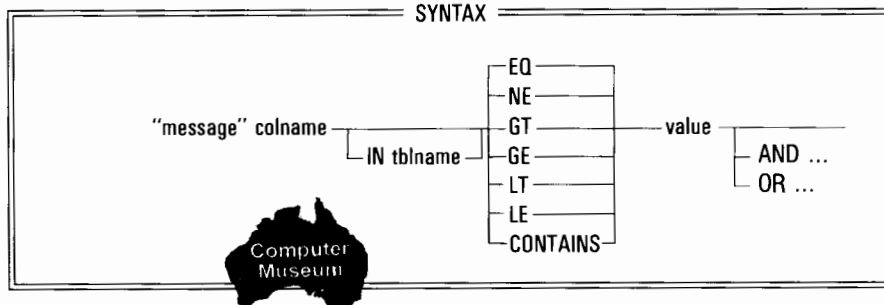
Table 6-3 Conditions for Defining Rules: Comparing a Column with a Value

Operator	Symbol	Definition
EQ	=	Equal
NE	< >	Not equal
GT	>	Greater than
GE	> =	Greater than or equal to
LT	<	Less than
LE	< =	Less than or equal to
CONTAINS	NONE	Contains a text value
EXISTS	NONE	Column contains data
FAILS	NONE	Column has a null value

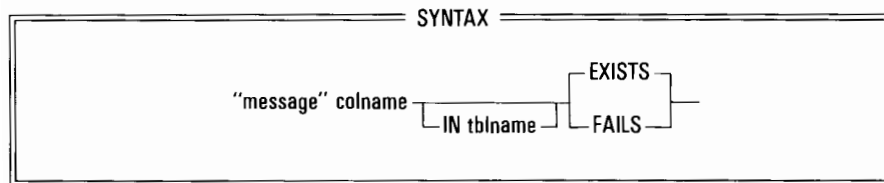
To enter rules that compare a column with a value, at the D> prompt type

# RULES

R:base responds with a D> prompt. Use the following syntax:



or



For example, suppose the customer identification number (*custid*) in COMPUCO's customer list (*custlist*) must be between 100 and 1000. Your entry to define the rule looks like this:

```

D>RULES
D>"invalid customer number" custid in custfile GE 100 AND custid IN custfile +
D>LE 1000
  
```

You can also define rules that compare the values of two columns. If the columns are in the same table, the rule compares column values in the same row. If the columns are in separate tables, the value of the first column defined in the rule is compared with every value of the second column. Table 6-4 lists conditions that you can specify to compare a column with another column.

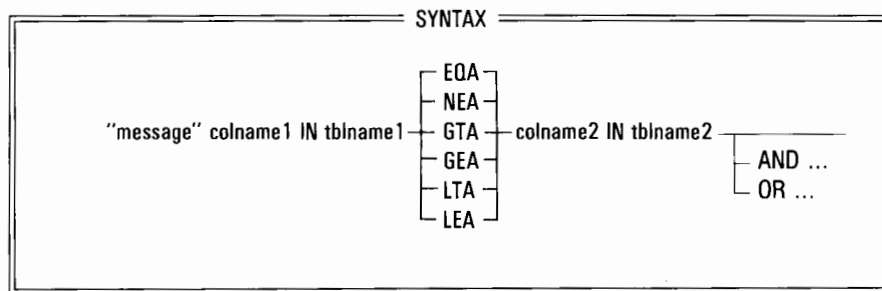
Table 6-4 Conditions for Defining Rules: Comparing Two Columns

Operator	Symbol	Definition
EQA	= A	Value of colname1 equals value of colname2
NEA	< > A	Value of colname1 does not equal value of colname2
GTA	> A	Value of colname1 is greater than value of colname2
GEA	> = A	Value of colname1 is greater than or equal to value of colname2
LTA	< A	Value of colname1 is less than value of colname2
LEA	< = A	Value of colname1 is less than or equal to value of colname2

To enter rules that compare a column with another column value, at the D> prompt type:

RULES

R:base responds with a D> prompt. Use the following syntax:



For example, suppose you do not want to issue the same product number to two COMPUCO products. The column *prodid* is in the table *product*. Your entry looks like this:

```
D> RULES
D> "duplicate product number" prodid IN product NEA prodid IN product
```

## Leaving the DEFINE Mode

The END command terminates the DEFINE mode. Enter END after all columns, tables, passwords, and rules have been defined. When you are ready to exit the DEFINE mode, at the D> prompt type:

```
END
```

A screen similar to the one shown below appears. Proceed with an R:base command or exit to the operating system.

```
D> END
END R:base Database Definition
R>
```

## REDEFINING THE DATABASE STRUCTURE

After your database is defined, you may find it necessary to change the structure. Columns may need additions, deletions, or complete redefinition, and tables may need expansion or removal. For example, suppose you define a column designed to contain company names as a TEXT data type with a length of 20 and later on you deal with a company with a name 30 characters long. The last 10 characters of the name will be cut off when the name is entered unless you redefine the column length as 30.

Tables also require expansion from time to time. Suppose you define a table that contains a column for company addresses and later on you deal with a company that has a street address for shipping and a post office box for billing. R:base enables you to rename the existing column as a shipping address and to add a mailing address column.

You can use the EXPRESS or the R:base COMMAND mode to redefine the database structure. Table 6-1 shows the tasks that you can accomplish with each method.

## Changing Column Definitions with CHANGE COLUMN

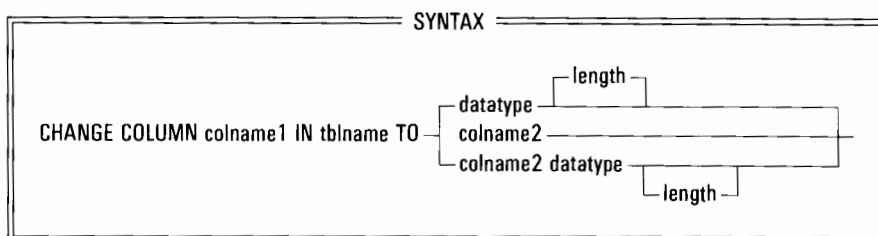
The CHANGE COLUMN command enables you to completely redefine a column. The column name, type, and length can all be changed or individually updated. If you change the data type, the data is converted to that type. For example, if a column data type is changed from REAL to INTEGER, the decimal values of the real numbers are truncated. If you change the data type of TEXT columns that contain text, all rows in the column are set to null.

You can change a column in all tables or in a specific table. If you change the data type of a column in one table and the column exists in another table, you must also rename the column with the CHANGE COLUMN command.

If you change a key column with the CHANGE COLUMN command, the new column is not a key. If you want the changed column to be a key column, you need to build a key for it using the BUILD KEY command that is explained later in this section. If you change a column that is used in *forms*, *reports*, *rules*, or command files, you must separately change the column in each of those tables or files.

If you change a column to a TEXT data type and do not assign a length, R:base assigns a length of eight characters. Before assigning lengths to other data types, read the precautions explained in "Step 3: Defining Tables."

To change a column, use the following format at the R > prompt:



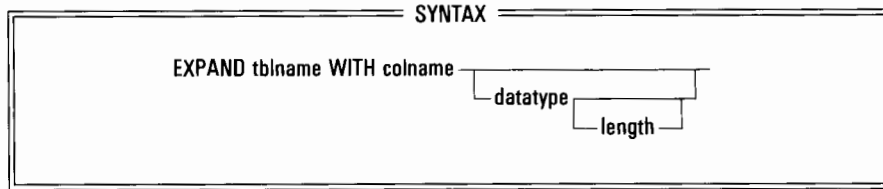
For example, suppose you need to change the *address* column in the table *custlist* to the name *shipadrs*. In addition, the length needs to be increased from 30 to 40 characters. The data type remains the same, but it must be entered so that the new length can be entered. At the R > prompt type:

```
CHANGE COLUMN address IN custlist TO shipadrs TEXT 40
```

R:base also has a CHANGE command that enables you to change the value of items in a column. That command pertains to modifying data stored in tables and is explained in chapter 9.

## Adding Columns to a Table with EXPAND

The EXPAND command enables you to add a new or existing column to an existing table. To expand a table, use the following format at the R> prompt:



For example, suppose you wish to add a mailing address column named *mailadr*s to the table *custlist*, at the R> prompt type:

```
EXPAND custlist WITH mailadr TEXT 40
```

If you change a column to a TEXT data type and do not assign a length, R:base assigns a length of eight characters. Before assigning lengths to other data types, read the precautions explained in “Step 3: Defining Tables.”

If you want the new column to be a key column, you need to build a key for it using the BUILD KEY command that is explained later in this section.

## Creating and Deleting Column Keys with BUILD/DELETE KEY

You can change the definition of columns from non-key to key, and from key to non-key. The column key is an optional entry for all data types. See “Step 2: Defining Column Names and Data Types” for additional information on column keys.

When you redefine a non-key column as a key, R:base establishes a data location index for the existing data in the column. The index is used and maintained as if the column had been declared key when the database was originally defined. To change a column from non-key to key, use the following format at the R> prompt:

```
BUILD KEY FOR colname IN tblname
```

For example, suppose you use the EXPAND command to add the column *mailadr*s to the table *custlist* and you want the column to be a key. At the R> prompt, type:

```
BUILD KEY FOR mailadr IN custlist
```

To change a column from key to non-key, use the following format at the R> prompt:

```
DELETE KEY FOR colname IN tblname
```

### Removing Columns from a Table with REMOVE COLUMN

The REMOVE COLUMN command deletes a column from an existing table. If the column is only located in one table, the structure and data for the column are deleted when the REMOVE COLUMN command is executed. If the column is located in more than one table, the command only deletes the column data from the specified table. To remove a column, use the following syntax at the R> prompt:

```
REMOVE COLUMN colname FROM tblname
```

For example, to remove the column *ext* from the table *salesrep*, at the R> prompt type:

```
REMOVE COLUMN ext FROM salesrep
```

### Removing Tables from a Database with REMOVE TABLE

You can also use the REMOVE TABLE command to remove a complete table—both the structure and data—from the open database. If a column in the deleted table only exists in that table, the information about the column's structure and data is also deleted. To remove a table, use the following format at the R> prompt:

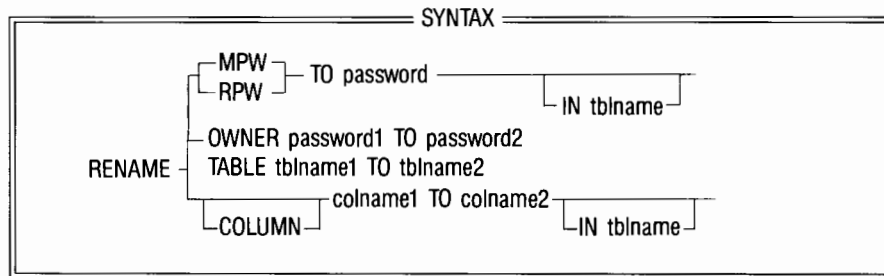
```
REMOVE tblname
```

For example, suppose you want to remove the table *forms* from the *compuco* database. Your entry and the R:base response are similar to the following screen. After you press [ENTER], R:base removes the table.

```
R>REMOVE forms
Press [ENTER] to remove the table forms      Press [ESC] to stop the command
```

## Renaming Columns, Passwords, and Tables with RENAME

The R:base RENAME command changes the name of a column, owner password, or table. You can rename a column in an entire database, or only in a single table. If you wish to rename a column throughout a database and the database is protected by an owner password, enter the owner password before using the RENAME command. If you rename a column that is used in *forms*, *reports*, *rules*, or command files, you must separately change the column name in each of those tables or files.



To rename a column, enter the old name and new name. For example, if you want to rename a column from *transid* to *transno* in the *transx* table, your entry and the R:base response look like this:

```
R> RENAME transid TO transno IN transx
Column transid renamed to transno in transx
R>
```

To rename an owner password, use the following format at the R> prompt:

```
RENAME OWNER password1 TO password2
```

For example, if you renamed the owner password from *opass* to *newpass*, your entry and the R:base response look like this:

```
R> RENAME OWNER opass TO newpass
Owner password opass      renamed to newpass
R>
```



To rename a modify password, use the following format at the R> prompt:

```
RENAME MPW TO password IN tblname
```

*IN tblname* is optional. The format to rename a read password is the same except you enter *RPW* instead of *MPW*.

To rename a table, use the following format at the R> prompt:

```
RENAME TABLE tblname1 TO tblname2
```

For example, if you want to change the name of a table from *custlist* to *custadrs*, your entry and the R:base response look like this:

```
R>RENAME TABLE custlist TO custadrs
Table custlist renamed to custadrs
R>
```

## Deleting Rules with DELETE

You may want to delete rules that no longer apply to the database. Use the LIST command to determine the rule number you wish to delete. Then use the DELETE command to remove the rule from the file. When you enter the DELETE command, *rules* is the table name. If you want to use a WHERE clause to pinpoint rows, use the column name *numrule* and enter the rule number. For example, if you wish to delete rules *six* and *seven*, at the R> prompt type:

```
DELETE ROWS FROM rules WHERE numrule EQ 6 OR numrule EQ 7
```

Additional information on the DELETE command is in chapter 9.

## DISPLAYING THE DATABASE STRUCTURE

The commands described in this section enable you to look at the structure of a database. You can use the LIST command to display:

- Databases in the current directory
- Table names in a database
- Every table in a database
- Columns in one table
- Rules in the database
- Forms in the database
- Reports in the database

## Displaying Database Names with LIST DATABASES

Use the LIST DATABASES command to display all the databases stored on the current drive and subdirectory. To enter the command, at the R> prompt type:

```
LIST DATABASES
```

## Displaying Table Names with LIST TABLES

The LIST TABLES command provides a complete list of the table names in the currently open database. If the database includes reports, forms, or rules, the tables *reports*, *forms*, or *rules* are listed. Reports, forms, and rules that you create with R:base are treated as rows within those tables.

For a list of all tables in the database *compuco*, your command and the R:base response look similar to this:

```
R>LIST TABLES
```

### Tables in the Database COMPUCO

Name	Columns	Rows	Name	Columns	Rows
transx	7	41	compnent	3	52
product	7	27	salesrep	10	5
custlist	9	9	compused	2	67
REPORTS	2	68			

## Displaying Table Structure with LIST

Use the LIST command to display the structure of every table in the database or only one table.

The LIST ALL command displays all of the columns in all the tables in a database. R:base lists the structure of each table one table at a time. For example, to list the structure of the tables in the database *compuco*, your entry and the R:base response look similar to this:

```
R>LIST ALL
```

```
Table: transx  
Read Password: NO  
Modify Password: NO
```

Column definitions

#	Name	Type	Length	Key
1	transid	INTEGER	1 value(s)	yes
2	empid	INTEGER	1 value(s)	
3	custid	INTEGER	1 value(s)	
4	prodid	TEXT	6 characters	
5	units	INTEGER	1 value(s)	
6	price	DOLLAR	1 value(s)	
7	tdate	DATE	1 value(s)	

```
Current number of rows: 41
```

```
More output follows - press [ESC] to quit, any key to continue
```

To list the structure of one table, enter the LIST command followed by a specific table name. The list includes the RPW/MPW password status, column number, data type, length, key status, and the current number of rows. For example, to list the structure of the table *transx*, your entry and the R:base response are similar to the following:

```
R>LIST transx
```

```
Table: transx
```

```
Read Password: NO
```

```
Modify Password: NO
```

```
Column definitions
```

#	Name	Type	Length	Key
1	transid	INTEGER	1 value(s)	yes
2	empid	INTEGER	1 value(s)	
3	custid	INTEGER	1 value(s)	
4	prodid	TEXT	6 characters	
5	units	INTEGER	1 value(s)	
6	price	DOLLAR	1 value(s)	
7	tdate	DATE	1 value(s)	

```
Current number of rows: 41
```

## Displaying Columns with LIST COLUMNS

The LIST COLUMNS command provides a detailed list of the column characteristics in the currently open database. The list includes the column name, data type, entry length, the table(s) where each column is used, and key status. For example, to list all of the columns in the database *compuco*, your command and the R:base response are similar to the following:

R> LIST COLUMNS

### Column definitions

Name	Type	Length	Table	Key
address	TEXT	30 characters	salesrep	
			custlist	
AND/OR	TEXT	4 characters	RULES	
BOOLEAN	TEXT	4 characters	RULES	
city	TEXT	20 characters	salesrep	
			custlist	
COLNAME1	TEXT	8 characters	RULES	
COLNAME2	TEXT	8 characters	RULES	
company	TEXT	40 characters	custlist	
compid	TEXT	6 characters	compnent	
			compused	
cost	DOLLAR	1 value(s)	product	
custid	INTEGER	1 value(s)	transx	
			custlist	
DATA	TEXT	80 characters	FORMS	
empid	INTEGER	1 value(s)	salesrep	

More output follows—press [ESC] to quit, any key to continue

## Displaying Rules with LIST RULES

The LIST RULES command provides a list of the data entry rules defined in the currently open database. The list includes the rule number and definition, and shows whether or not the R:base RULE CHECKING command is set on or off. To display rules, use the following syntax at the R> prompt:

SYNTAX	
LIST RULES	<input type="text" value="rule number"/>

Use the *rule number* option to request a specific rule.

To review the rules for the *compuco* database, your command and the R:base response look similar to this:

```
R>LIST RULES
```

```
    RULE checking = ON
    RULE 1 custid IN custlist GE 100 AND custid IN custlist LE 1000
      Message:invalid customer number
    RULE 2 prodid IN product NEA prodid in product
      Message:duplicate product number
```

```
R>
```

If you only want to look at the second rule, at the R> prompt type:

```
LIST RULES 2
```

## Displaying Form Names with LIST FORMS

The LIST FORMS command displays all the form names in the database. To display a list of form names, at the R> prompt type:

```
LIST FORMS
```

## Displaying Report Names with LIST REPORTS

The LIST REPORTS command displays all the report names in the database. For a list of report names, at the R> prompt type:

```
LIST REPORTS
```



---

## Data Entry Contents

<b>How to Use This Chapter</b>	7-2
<b>Data Entry Methods</b>	7-2
<b>Creating Data Entry Forms</b>	7-2
Creating Table Forms	7-3
Creating Variable Forms	7-3
Beginning the Form Definition	7-3
Editing a Form	7-5
Locating Columns	7-7
Changing the Data Entry Order of a Form	7-11
Ending the Form Definition	7-11
<b>Maintaining Forms in a Database</b>	7-12
Listing All the Form Names in the Database	7-12
Displaying the Layout Specifications of Forms	7-12
Deleting Forms	7-13
<b>Entering Data</b>	7-14
Data Entry Using Table Forms	7-14
Data Entry Using Variable Forms	7-16
Data Entry Using the LOAD Command	7-16
Loading Data with Prompts	7-16
Loading Data without Prompts	7-17
Data Entry Commands Used with the LOAD Command	7-19



## HOW TO USE THIS CHAPTER

This chapter explains the procedures for entering data into a database with the ENTER and LOAD commands. Before you can enter data, you must first build a database as explained in chapter 6. Most of this chapter is a guide to creating and maintaining data entry forms, and entering data with the forms. The last section explains how to enter data without forms.

Data entry using the INPUT command is explained in chapter 12. If you want to enter data that was created with another program, refer to chapter 13.

## DATA ENTRY METHODS

You can use two commands to enter data into an R:base database, ENTER and LOAD. The ENTER command requires that you first define a table form.

The FORMS mode enables you to design custom data entry screens. You can use the FORMS command to create a data entry form for a specific table or for an application. Defining a form enables you to develop on-line headings and messages that clarify the data entry process. After you define a form, you can also use it with the EDIT command to modify information in the database.

The LOAD command does not require predefined forms and works independently of any forms that you have defined. The LOAD command offers an alternative to entering data with a form and can be used to enter data from an existing file.

## CREATING DATA ENTRY FORMS

R:base enables you to make two kinds of data entry forms. Table forms are easy to define without previous experience and are used to add or edit data in a single table. Variable forms require experience in building command files and are used to create multiple-page forms, forms that load or edit data located in more than one table, and other similar applications. Both types of forms allow you to design a custom display for data entry. You can define heading names, footnotes, and messages that apply to an entry task, and locate each on the screen.

Although the screen definition process is similar for defining table and variable forms, a form created under one process cannot be used by the other process. For example, if you define a table form, you cannot use it to load data using variables. Do not attempt to define a variable form unless you have read about variable form applications in chapter 15.

## Creating Table Forms

A table form is a single screen data-entry form that is easy to design. You can place headings and explanatory notes anywhere on the screen and locate the areas where the entered data is displayed. Each table form relates to a specific table in the database. The table must exist before you can define a corresponding form. Not all columns from the table have to be included on the form. If you only use some of the columns, the columns that are not listed are assigned null values when the form is used for data entry.

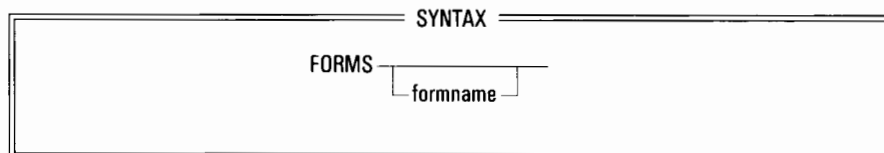
## Creating Variable Forms

A variable form is similar to the basic table form. Like the table form, you can position headings and explanatory notes anywhere on the screen and locate the areas where the entered data is displayed. Unlike table forms, a variable form is not associated with a specific table. You can use one form to load several different tables, or you can package several single-page forms together and display them as one multiple-page form. You can control the prompts that operators will see, and set up different routines for various levels of users.

Variable forms are executed by commands in an R:base command file, and as a result, the functionality is dependent upon your programming knowledge. Chapter 15 explains how to build a variable forms application. This chapter focuses only on table forms.

## Beginning the Form Definition

You start defining a form by entering the FORMS command at the R> prompt. Enter the command with a form name. Form names must be from one to eight characters. The name must begin with a letter, and cannot contain spaces or special characters such as hyphens or colons. If you enter an existing form name, you can edit the existing form. If you enter a new form name, you can define a new form. The syntax for the command is:



After you enter the command, a message confirms that you are defining a form. If you are modifying an existing form, the FORMS menu is displayed above the form. If you are defining a new form, you are asked for the table that the form is associated with. For example, if you are defining a new table form named *transx*, your command and the FORMS response are similar to the following:

```
R>FORMS transx
Begin R:base forms definition
Enter table name (for variables form press [ENTER] ):
```

Enter the name of the table the form is associated with. The FORMS menu is displayed, giving you the option to *Edit*, *Locate*, or *Quit*.

If you press the [ENTER] key instead of entering a table name, FORMS automatically begins defining a variable form and responds with the message:

```
Defining VARIABLES form
Press [ESC] to quit, any key to continue
```

If you press [ESC], you are returned to an R > prompt. If you press any other key, the FORMS menu is displayed. Use the right-arrow and left-arrow keys to move the cursor to the option you want and press [ENTER], or type the first letter of the option you want.

- *Edit* allows you to enter the headings and messages you want to include on the form. See the “Editing a Form” section.
- *Locate* allows you to specify the areas where the data is to be entered. See the “Locating Columns” section.
- *Quit* allows you to leave the FORMS definition mode. See the “Ending the Form Definition” section.

## Editing a Form

Choose the *Edit* option from the FORMS menu to type the headings and messages you want to include on your form. If you are creating a new form, the screen will be blank except for the following message in the upper right corner:

< 1, 1> [F3] to list, [ESC] to exit

The numbers in the angle brackets (< >) refer to the cursor location. The first number identifies the line and the second number identifies the column. To list the columns contained in the table, press [F3]. To exit to the FORMS menu, press [ESC].

If you are editing an existing form, the form appears on the screen with the message shown above in the upper right corner.

The design of the form is completely open. Any part of the screen can be used as a heading, message, or data entry area. The following guidelines also apply. The guidelines refer to columns and column names, but also apply to variables and variable names.

- The form can be no larger than the display screen: The maximum width of the form is 80 characters, and the maximum length is 23 rows.
- A column heading on the form can be given a different name than the column name. For instance, if the column name is *firstname*, you can assign it the heading FIRST NAME.
- Column values can be placed together on the same line of a form. For example, if one column contains first names and another column contains last names, they can be located next to each other so that the entire name is entered on one line. Note that although the entries are on the same line, the first and last names are stored in separate columns.
- Special characters may be included when a form is designed. Refer to your operating system manual to determine the series of keystrokes required to create special characters. FORMS does not accept all special characters, but many of the characters used to create boxes and other common graphic effects can be used.

When you edit a form, use the special function keys listed in table 7-1.

Table 7-1 Special Function Keys Used in the FORMS Editing Mode

Key(s)	Function
[Tab]	Moves the cursor ten spaces to the right
[Shift] [Tab]	Moves the cursor ten spaces to the left
[→]	(Cursor right) Moves the cursor one space to the right
[←]	(Cursor left) Moves the cursor one space to the left
[↑]	(Cursor up) Moves the column cursor up one line
[↓]	(Cursor down) Moves the column cursor down one line
[Ctrl] [→]	Moves the cursor to the end of the line
[Ctrl] [←]	Moves the cursor to the beginning of the line
[Del]	Removes the character at the current cursor position
[Ins]	Inserts a space at the current cursor position
[F1]	Inserts a blank line at the current cursor position and moves all following lines down one line
[F2]	Deletes the line at the current cursor position and moves all following lines up one line
[F3]	Lists the columns in the table
[F4]	Toggles the REPEAT mode ON and OFF. In the REPEAT mode, the last character entered is repeated when the cursor is moved. When REPEAT is displayed on the top line of the screen, the REPEAT mode is on

When you design a form, you can tailor the screen to fit your needs. For example, suppose you work for **COMPUCO** and you are asked to design a table form for use by the sales representatives. The form you need to design corresponds with the sales transaction table *transx*, and you decide to name it *trnsxfrm*.

At the R> prompt, type:

```
FORMS trnsxfrm
```

When you are asked for a table name, type:

```
transx
```

FORMS displays a screen that is blank except for the notes in the upper right hand corner. Type the headings and messages you want on the form. If you have forgotten the columns that are in the table, press [F3]. When you type information on the screen, be sure to leave enough room for the entered data. When you finish, your screen might look like the one below.

< 1, 1 > [F3] to list, [ESC] to exit

Transaction number (Be sure to clear this with Ruth!)

Your employee number (Pete-133 Ernie-129 Mary-160 John-131)


Customer ID number (Listed in table called custlist)

What did you sell? (List the product number)

How many units did you sell?

What was the price per unit?

What is the transaction date?



After you finish designing your form, press the [ESC] key. The FORMS menu appears. If you have entered everything correctly, press [L] and proceed with locating the data entry areas for columns or variables. If you need to modify the form, press [E].

## Locating Columns

Choose the *Locate* option from the FORMS menu to define the data entry areas for columns. After you press the [L] key, a message similar to the following appears above the form:

Enter column name : [F3] to list [ESC] to exit

Enter the name of the column. Instead of typing the name, you can enter

#*c*

where *c* is the item number that is shown when you press [F3] to list the columns. Note that the order in which you locate columns on the form defines the order in which you enter data.

If you are locating columns on a new form, or if you are relocating a column, FORMS displays a message similar to the following:

Move cursor to start location for COLNAME and press [S] < 1, 1>

Place the cursor where you want the data entry area to begin and press [S]. FORMS indicates the beginning of the area with an *S* and displays a message similar to the following:

Move cursor to end location for COLNAME and press [E] < 1, 45>

FORMS automatically moves the cursor to the maximum length position of the item. If you like, you can reduce the length of areas for INTEGER, REAL, and TEXT data types. Move the cursor to the place on the form where you want the data entry area to end and press [E]. FORMS indicates the end of the area with an *E* and requests another column or variable name.

Continue to enter locations until the entire form is complete. Press [F3] if you need help remembering the columns in the table. If you decide you want to change the location of an item, all you need to do is reenter the item name. When you have located all the items, press [ESC] to return to the FORMS menu.

Returning to the COMPUCO transaction form defined in the previous section, you can now locate the data entry areas as explained below.

When the FORMS menu is displayed, press [L] to locate the columns. The form *trnsxfrm* is displayed and you are requested to enter a column name. Enter the column name for the transaction number, *trnsx*. The screen looks like this:

Move cursor to start location for TRANSX and press [S]

< 1, 1 >

Transaction number (Be sure to clear this with Ruth!)

Your employee number (Pete-133 Ernie-129 Mary-160 John-131)

Customer ID number (Listed in table called custlist)

What did you sell (List the product number)

How many units did you sell?

What was the price per unit?

What is the transaction date?

Move the cursor to the right of the end parenthesis in the *Transaction number* line and press [S]. FORMS marks the beginning location with an S and asks you to locate the end point of the transaction number entry area. FORMS automatically moves the cursor to the maximum length for the item. Note that TEXT columns longer than 80 characters are located in one block. FORMS starts at the beginning location and proceeds from left to right. When the right edge of the screen is reached, the area is continued on the next line at column 1.



Press [E] when the cursor is where you want it. When FORMS requests another column name, enter the next name, *empid*, and continue locating entry areas. When you are done locating areas, your screen looks similar to this:

```

Enter column name :                               [F3] to list [ESC] to exit
Transaction number (Be sure to clear this with Ruth!) S      E
Your employee number (Pete-133 Ernie-129 Mary-160 John-131) S  E
Customer ID number (Listed in table called custlist) S      E
What did you sell? (List the product number) S      E
How many units did you sell? S      E
What was the price per unit? S      E
What is the transaction date? S      E

```

If you are editing the location of an existing column, FORMS highlights the current data entry area for the item and displays a menu that looks like this:

```

--Keep--Set--Change--Delete-----Location for: colname

```

- Press [K] if you want to save the current location. FORMS saves the location and asks you for another column or variable name.
- Press [D] if you want to remove the entry area from the form. FORMS deletes the area and asks you for another column or variable name.
- Press [S] or [C] if you want to change the location. FORMS erases the previous location and asks you to relocate the area.

When you are finished locating column areas, press [ESC] to return to the FORMS menu.

## Changing the Data Entry Order of a Form

After you finish defining a form, you may want to change the order in which data is entered. Choose the *Locate* option from the FORMS menu. The form is displayed and you are asked to enter a column name. Enter the name of the first column for which you want to enter data. FORMS displays the column and you are given the option to *Keep*, *Set*, *Change*, or *Delete* the current start and end locations of the data entry area. Choose the *Delete* option. FORMS asks you for another column name. Reenter the name of the first column for which you want to enter data and locate the data entry area. Perform this procedure for each column on the form. If you do not relocate every column in the table, the columns that you relocate are last in the entry order. When you are finished, choose the *Quit* option from the FORMS menu and the *Save changes* option from the following menu.

## Ending the Form Definition

Choose the *Quit* option from the FORMS menu to complete the form definition. You are given the option to *Save changes*, *Discard changes*, or *Return*.

When the menu is displayed, use the space bar to move the cursor to the option you want and press [ENTER], or press the first letter of the option you want.

- *Save changes* stores the form that you just finished creating or editing and returns to an R> prompt.
- *Discard changes* stores the form that was originally displayed when you entered the form name. If you made any changes to the form, the changes are not recorded. If you just created a new form, choosing the *Discard* option deletes the form from memory. After the information is discarded, you are returned to an R> prompt.
- *Return* directs FORMS to display the FORMS menu. You can edit the form, locate new columns, relocate existing data entry areas, or quit the definition process.

When you save a form in the FORMS definition mode and return to an R> prompt, the R:base message looks like this:

```
Storing form
End R:base forms definition
R>
```

## MAINTAINING THE FORMS IN A DATABASE

### Listing All the Form Names in the Database

If you want a listing of the form names that have been defined in the open database, at the R> prompt type:

```
LIST FORMS
```

The command displays a list of the form names and the tables associated with them.

For example, if you enter the LIST FORMS command, a screen similar to the one shown below appears. For additional information on the LIST command, see chapter 5.

```
R> LIST FORMS
FORM      TABLE
-----
trnsxfrm  transx
```

### Displaying the Layout Specifications of Forms

All the forms that you create are stored in a table called *forms*. To display the layout specifications for all the defined forms, at the R> prompt type:

```
SELECT ALL FROM FORMS
```

Note that you can also use other R:base commands to edit or change the data in the *forms* table. The SELECT command displays all the rows in the *forms* table. The form name is displayed beneath the column heading FNAME, and layout information is located beneath the column heading FDATA. The order of the information beneath the row that specifies LAYOUT is:

- form name
- column name
- line where the data entry area starts
- column where the data entry area starts
- column length

For example, if you want to list the specifications of the form *trnsxfrm* in the *compuco* database, your entry and the R:base response look similar to the following screen. For additional information on the SELECT command, see chapter 8.

```
R>SELECT ALL FROM FORMS WHERE FNAME EQ trnsxfrm
```

```
FNAME  FDATA
```

```
-----
trnsxfrm transx
trnsxfrm
trnsxfrm Transaction number (Be sure to clear this with Ruth!)
trnsxfrm
trnsxfrm Your employee number (Pete-133 Ernie-129 Mary-160 John-131)
trnsxfrm
trnsxfrm Customer ID number (Listed in table called custlist)
trnsxfrm
trnsxfrm What did you sell? (List the product number)
trnsxfrm
trnsxfrm How many units did you sell?
trnsxfrm
trnsxfrm What was the price per unit?
trnsxfrm
trnsxfrm What is the transaction date?
trnsxfrm LAYOUT
trnsxfrm transid      2  55  10
trnsxfrm empid       4  62  10
trnsxfrm custid      6  54  10
trnsxfrm prodid      8  46   6
```

```
More output follows—press [ESC] to quit, any key to continue
```

## Deleting Forms

When a form is no longer needed, you can remove it using the `DELETE` command. Each form in the *forms* table is a collection of rows, and the form names are stored in the column *fname*. For example, if you have created two forms, one named *trnsxfrm* and one named *custlist*, the *forms* table has several rows where *fname* is *trnsxfrm* and several rows where *fname* is *custlist*. To delete an entire form, use the following syntax:

```
DELETE ROWS FROM FORMS WHERE FNAME EQ formname
```

For example, if you want to remove *trnsxfrm* from the *forms* table, your entry and the R:base response look like this:

```
R>DELETE ROWS FROM FORMS WHERE FNAME EQ trnsxfrm
45 row(s) have been deleted from FORMS
```

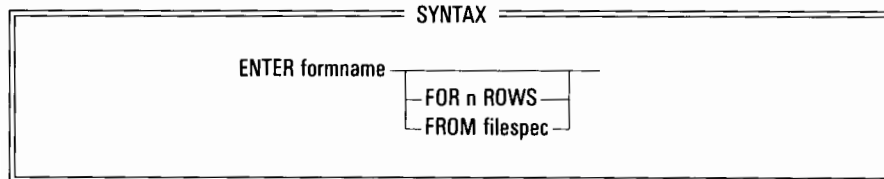
## ENTERING DATA

You can use two basic data entry methods within R:base. You can enter data with the ENTER command and a previously defined form, or with the LOAD command.

R:base checks all data entries to make sure that they are the correct data type and do not exceed the maximum length. If the data type is wrong or if the entry is too long, an error message indicates the problem. When an invalid data type is entered, the information is not added to the database. If you have defined rules for your database, R:base also verifies all entries with the rules unless you enter the SET RULES OFF command to turn off rule checking. For additional information on rules, see chapter 6.

## Data Entry Using Table Forms

Use the ENTER command when you want to enter data using a table form. Use the following syntax to enter data with a table form:



The option *FOR n ROWS* determines how many rows of data (*n*) may be entered. For example, if *n* is set equal to one (FOR 1 ROW), the form is displayed once and you are able to enter one row of data. After the entry you are automatically returned to the R> prompt or to the next command line in a command file. This feature is convenient for applications that need to process other kinds of information after each entry.

The option *FROM filespec* enables you to enter data directly from an existing formatted ASCII file. First, you must define a table that has all the columns that the ASCII file contains. Then you must make a corresponding form. See the "Loading Fixed Field ASCII Files into R:base" section of chapter 12 for additional information.

After you execute the ENTER command, FORMS clears the screen and displays your form. A reverse video bar highlights the first column that you located when you defined your form, and the following message appears at the top of the screen:

Press [ESC] when done with this data

For example, if you typed the ENTER command for the sample COMPUCO form shown in this chapter, your entry looks like this:

ENTER trnsxfrm

The FORMS response looks like this:

Press [ESC] when done with this data

Transaction number (Be sure to clear this with Ruth!)

Your employee number (Pete-133 Ernie-129 Mary-160 John-131)

Customer ID number (Listed in table called custlist)

What did you sell? (List the product number)

How many units did you sell?

What was the price per unit?

What is the transaction date?

If you first enter the command SET AUTOSKIP ON, the cursor moves automatically to the next entry column as soon as all spaces in the current column are filled. See chapter 5 for more information.

After data for each column in the row has been entered, press [ESC]. A menu that looks like this is displayed:

--Add--Reuse--Edit--Quit-----

Use the right-arrow and left-arrow keys to move the cursor to the option you want and press [ENTER], or type the first letter of the option you want.

- If the information you loaded does not require changes, press [A] to add a new row to your table. FORMS loads the row and displays a blank entry form.
- If the data entered requires changes, press [E] to edit the information. FORMS redisplay the row that you just entered.
- If the entry does not require changes, and is very similar to another entry you need to make, press [R]. FORMS adds the row to the table and displays the same data on the screen. If you are entering repetitive information into a table, use this option to save time.
- If you do not want to load the row, press [Q]. If you want to load the last row entered, be sure to press [A] before you press [Q].

## **Data Entry Using Variable Forms**

Variable forms are developed as part of an application using the R:base programming language. With variable forms, you, or someone you hire, creates an application to accomplish complex data loading tasks. For additional information on building an application to use with a variable form see chapter 15.

## **Data Entry Using the LOAD Command**

The LOAD command enables you to enter data without defining an entry form, and can be used with or without system prompts.

### **LOADING DATA WITH PROMPTS**

When you execute the LOAD command using system prompts, R:base presents the data on the screen in a column-oriented format, displaying the column name and data type as a prompt.

When you load data with prompts, you load one row of data at a time. If an entry is the wrong data type an error message is immediately shown. If an entry does not conform with the data entry rules that you defined, the rule error message is shown after the entire row is entered.

The maximum text entry length when you load data with prompts is 80 characters. When you want to enter more than 80 characters, load the data without prompts or make a custom data entry form.

To enter data using prompts, use the following syntax:

SYNTAX	
LOAD tblname WITH PROMPTS	— USING collist —

The *USING collist* option specifies the columns to be loaded, and the column loading order. All columns in the table that are not included in the column listing are assigned null values when the rows are entered. For example, if you want to load the *transid*, *empid*, and *price* columns in the table *transx*, your entry looks like this:

LOAD transx WITH PROMPTS USING transid empid price

After you press [ENTER] to begin data loading, the first data entry screen looks similar to this:

Press [ESC] to end [RETURN] to continue  
TRANSID (INTEGER):

After you enter data for *transid* and press [ENTER], you are prompted for the next column entry, *empid*. If you do not want to load the row that you are currently entering, press [ESC] at any time.

#### LOADING DATA WITHOUT PROMPTS

When data is loaded without prompts, there is no on-line assistance to help you remember the entry order of the columns. You must specify the order of data entry with a *USING* clause or follow the order in which the columns were listed when the table was defined.

To load data without prompts, the syntax is:

SYNTAX	
LOAD tblname	— USING collist —



The option *USING collist* enables you to enter data for selected columns in the table and to specify the loading order. All columns in the table that are not included in the column listing are assigned null values when the rows are entered.

The LOAD command also enables you to enter data directly from an existing ASCII file. If you want to use the LOAD command to load data from a file, see chapter 12 for information.

After you enter the LOAD command, R:base responds with an L> prompt. For example, if you want to load the table *transx*, your entry and the R:base response look like this:

```
R>LOAD transx
Begin R:base Data Loading
L>
```

To load the data, you must remember the order in which the columns were defined or follow the order specified with the USING option. Separate the entries with commas or a blank space. If TEXT data type entries contain blank spaces, plus or equal signs, or semicolons, enclose the entry in quotation marks. If TEXT data type entries contain commas or quotation marks, see the “Setting System Default Values” section in chapter 5.

In the example table *component*, the loading order is: *compid*, *proddesc*, and *prodname*. A typical entry looks like this:

```
L> 1000 "Basic Keyboard" "standard keyboard component"
```

At the end of each line, press [ENTER].

You can also load global variable values with the LOAD command. For example, if you are loading the *tdate* column in the table *transx*, you can use the variable *#date* to enter the current date. Precede the variable name with a period. The loading order for *transx* is: *transid*, *empid*, *custid*, *prodid*, *units*, *price*, and *tdate*. Your entry looks like this:

```
L> 4790,129,105,CX3000,15,1900.00, #date
```

When you are finished loading data, at the L> prompt type:

END

Your entry and the R:base response look like this:



```
L>END
End R:base Data Loading
R>
```

If R:base responds with a +> prompt, one of your entries is missing a quotation mark ("). To return the system to an R> prompt, enter a quotation mark (") and press [ENTER]. Then check the data you just loaded to correct any input errors that may have occurred.

### Data Entry Commands Used with the LOAD Command

There are two sets of R:base commands that can be used with the LOAD command: FILL and NOFILL, and CHECK and NOCHECK.

The FILL command directs R:base to enter a null value for columns that have not been assigned a value. If a rule specifies that a column requires an entry other than null, an error message is displayed and the row is not loaded.

To enter the FILL command, at the L> prompt type:

FILL

For example, if you are loading a table that has five columns but you presently only have information for the first three, you can activate the FILL command to automatically enter null values for the last two columns. Instead of typing:

```
L>100,200,250,-0,-0-
```

you can enter the FILL command and type:

```
L>FILL
L>100,200,250
```

The FILL command assigns null values to the last two columns. The command only fills in values that are missing at the end of the row. If you want to assign null values to items in the middle of a row, you have to enter the null symbol (-0-) instead of a value.

The NOFILL command turns off the FILL command. When you enter the NOFILL command, R:base will not accept data for a row unless all columns have been assigned a value. If you entered the FILL command, it is not necessary to enter NOFILL when you finish loading data because the R:base default value is NOFILL. To enter the NOFILL command, at the L> prompt type:

NOFILL

When you are in the LOAD mode (L>), the CHECK and NOCHECK commands direct R:base to verify or not verify data entries with the database rules you defined. If an entry does not satisfy the conditions of a rule, the error message you created for that rule is displayed and the entry is not accepted. CHECK directs R:base to check input against all existing rules. NOCHECK directs R:base to ignore all existing rules. The R:base default value is CHECK.

When you do not want to verify data input with the database rules, at the L> prompt type:

NOCHECK

When you want to verify data entries, at the L> prompt type:

CHECK

If you have defined an owner password for your database, R:base does not accept the NOCHECK command until you enter the password at the R> prompt.

---

## Data Inquiry Contents

<b>How to Use This Chapter</b>	8-2
<b>Viewing the Database Contents</b>	8-2
Using the SELECT Command	8-2
Specifying Column Display Widths	8-5
Computing Column Totals	8-7
<b>Performing Arithmetic Calculations</b>	8-8
Using the COMPUTE Command	8-8
<b>Tallying Values in a Column</b>	8-10
Using the TALLY Command	8-10

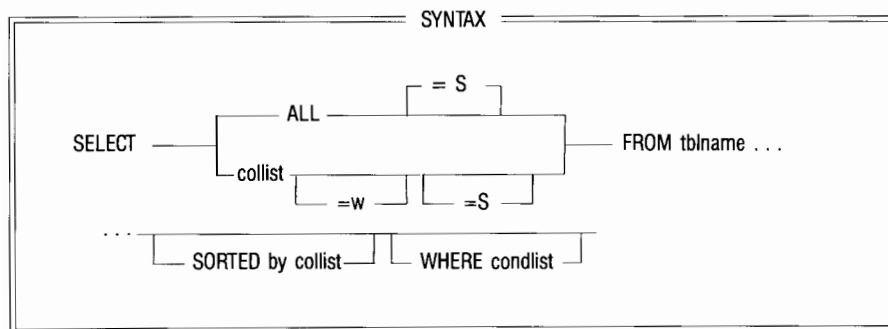
## HOW TO USE THIS CHAPTER

This chapter explains how you can display the data that you have entered in your database. The three R:base data inquiry commands are: **SELECT**, **COMPUTE**, and **TALLY**. All three commands can be used in the **PROMPT** mode.

## VIEWING THE DATABASE CONTENTS

### Using the **SELECT** Command

You can use the **SELECT** command to retrieve information from the database in several different ways. The syntax for the command looks like this:



SELECT ALL retrieves all of the data in a table. If there are too many columns to fit on the screen, R:base shows you as many of the columns as it can. The following screen is an example of the SELECT command used to view all of the data stored in the table *product*. The table is located in the database *compuco*. *Product* consists of seven columns, but note that R:base is only able to display the first three on an 80 character screen. When you direct output to a printer or file, you can set the width as wide as 256 characters. For more information on setting the width, see the "Setting Special Characters" section in chapter 5.

```
R> SELECT ALL FROM product
```

prodid	proddesc	prodname
CX3000	System-256K-Dual Drive	Standard Color PC
CX3010	System-512K-Dual Drive	Classic Color PC
CX3020	System-Dual Drive w/Hard Disk	Advanced Color PC
CX3030	System-Single Drive w/Hard Disk	Superior Color PC
MB3000	System-512K-Dual Drive	Standard PC
MB3020	System-Dual Drive w/Hard Disk	Advanced PC
MB3030	System-Single Drive w/Hard Disk	Superior PC
MX3000	System-256K-Dual Drive	Standard PC with deluxe keyboard
MX3010	System-512K-Dual Drive	Classic PC with deluxe keyboard
MX3020	System-Dual Drive w/Hard Disk	Advanced PC with deluxe keyboard
MX3030	System-Single Drive w/Hard Disk	Superior PC with deluxe keyboard
PB3040	System-256K-Dual Drive-Portable	Portable Standard PC
PB3050	System-512K-Dual Drive-Portable	Portable Classic PC
PX3040	System-256K-Dual Drive-Portable	Portable Standard PC w/deluxe kbd
PX3050	System-512K-Dual Drive-Portable	Portable Classic PC w/deluxe kbd

SELECT *colist* retrieves only selected columns. You can select up to 40 columns and list them in any order. The current width setting, however, restricts the number of columns that are actually displayed. The following screen is an example of the SELECT command used to view the data for the columns *prodid*, *listprce*, *cost*, and *prodname* in the table *product*.

```
R>SELECT prodid listprce cost prodname FROM product
```

prodid	listprce	cost	prodname
CX3000	\$1,900.00	\$1,530.00	Standard Color PC
CX3010	\$2,275.00	\$1,730.00	Classic Color PC
CX3020	\$2,575.00	\$1,930.00	Advanced Color PC
CX3030	\$2,875.00	\$2,130.00	Superior Color PC
MB3000	\$1,800.00	\$1,400.00	Standard PC
MB3020	\$1,975.00	\$1,800.00	Advanced PC
MB3030	\$2,350.00	\$2,000.00	Superior PC
MX3000	\$1,600.00	\$1,430.00	Standard PC with deluxe keyboard
MX3010	\$1,875.00	\$1,630.00	Classic PC with deluxe keyboard
MX3020	\$2,300.00	\$1,830.00	Advanced PC with deluxe keyboard
MX3030	\$2,575.00	\$2,030.00	Superior PC with deluxe keyboard
PB3040	\$2,500.00	\$2,175.00	Portable Standard PC
PB3050	\$3,000.00	\$2,300.00	Portable Classic PC
PX3040	\$3,100.00	\$2,225.00	Portable Standard PC w/deluxe kbd
PX3050	\$3,175.00	\$2,350.00	Portable Classic PC w/deluxe kbd

You can use the SELECT command with a table name in conjunction with two clauses: SORTED BY and WHERE. The SORTED BY clause allows you to choose which row is listed first, and whether the rows are listed in ascending or descending order. The WHERE clause is used to narrow the amount of information displayed by requesting data that fits a certain definition. See chapter 5 for a more complete discussion of the SORTED BY and WHERE clauses.

You can also use a route number with the SELECT command, however, the route only points to the first row. The SELECT then displays all rows following the first row pointed to not just the rows selected by the SET POINTER command. See information on the SET POINTER command in chapter 15 and in the *R:base 5000 Reference Manual*.

## Specifying Column Display Widths

R:base allows you to adjust the display width of columns. Table 8-1 shows the default display value for each data type. You can decrease the width for DOLLAR data types or increase the width to as many as 23 spaces. You cannot increase the width for the other three types, but you can decrease it. For example, R:base allots 10 spaces for INTEGER columns. If you always display values smaller than 999999, you can specify a width of six spaces.

Table 8-1 Display Width Default Values

Text Type	Default Value (Number of Characters)
DATE	8 to 11 depending upon the format selected
DOLLAR	15
INTEGER	10
REAL	8
TEXT	Length specified when the column was defined
TIME	8

If you wish to adjust the display width of the column, specify the number of characters to be displayed immediately after the column name. R:base displays TEXT type data regardless of the column width you choose. However, displays of all other data types require a column width large enough to display the entire item on one line. Asterisks are displayed if there is not enough room to display the item. To specify column widths, use the following syntax:

### SYNTAX

```
SELECT colname1=w colname2=w FROM tblname
```



The following screen is an example of a reduced column width display from the *product* table in the *compuco* database. See the previous screen for an illustration of how R:base displays the first four columns when no widths are specified.

```
R> SELECT prodid listprce=10 cost=10 prodname=18 proddesc=10 FROM product
```

prodid	listprce	cost	prodname	proddesc
CX3000	\$1,900.00	\$1,530.00	Standard Color PC	System-256 K-Dual Drive
CX3010	\$2,275.00	\$1,730.00	Classic Color PC	System-512 K-Dual Drive
CX3020	\$2,575.00	\$1,930.00	Advanced Color PC	System-Dua l Drive w/Hard Disk
CX3030	\$2,875.00	\$2,130.00	Superior Color PC	System-Sin gle Drive w/Hard Disk
MB3000	\$1,800.00	\$1,400.00	Standard PC	System-512 K-Dual Drive
MB3020	\$1,975.00	\$1,800.00	Advanced PC	System-Dua l Drive w/Hard Disk

## Computing Column Totals

The SELECT command also has an option that allows you to compute column totals for DOLLAR, INTEGER, and REAL data types. Specify column totals (=S) immediately after the column name in the SELECT command. The totals are displayed after the last row in the table. If you wish to display all columns in a table and totals for all DOLLAR, INTEGER, and REAL data types, use the following syntax:

### SYNTAX

```
SELECT ALL=S FROM tblname
```

R:base cannot compute column totals for data types other than DOLLAR, INTEGER, and REAL. If you want to display totals for selected columns, use the following syntax:

### SYNTAX

```
SELECT colname1=S colname2=S colname3 colname4 FROM tblname
```

The following screen shows the summation of the *units* and *price* columns in the table *transx*. All the columns are displayed, but because the only meaningful totals are for *units* and *price*, the other columns are not totaled.

```
R>SELECT transid empid custid prodid units=S price=S FROM transx
```

transid	empid	custid	prodid	units	price
4790	129	105	CX3000	15	\$1,900.00
4791	129	105	CX3010	25	\$2,275.00
4792	129	102	CX3020	5	\$2,575.00
4793	131	104	CX3020	12	\$2,575.00
4794	102	101	CX3030	50	\$2,875.00
4795	102	101	MB3000	10	\$1,800.00
4800	129	105	MB3000	25	\$1,600.00
4865	102	102	MB3020	5	\$1,975.00
4870	129	105	MB3020	25	\$1,975.00
4874	102	102	MB3030	5	\$2,350.00
4875	102	101	MX3000	50	\$1,600.00
4796	133	100	PB3050	8	\$3,000.00
				235	\$26,500.00

You can specify column widths and compute column totals with one command line. When you want to perform both operations, list the compute column total (=S) after the width specification (=w). For example, suppose you want to look at the columns *prodid*, *listprce*, *proddesc*, and *prodname* in the table *product* and you also want to total the *listprce* column. At the R> prompt type:

```
SELECT prodid listprce=10=S proddesc=10 prodname FROM product
```

For additional information on specifying column widths, see the previous section.

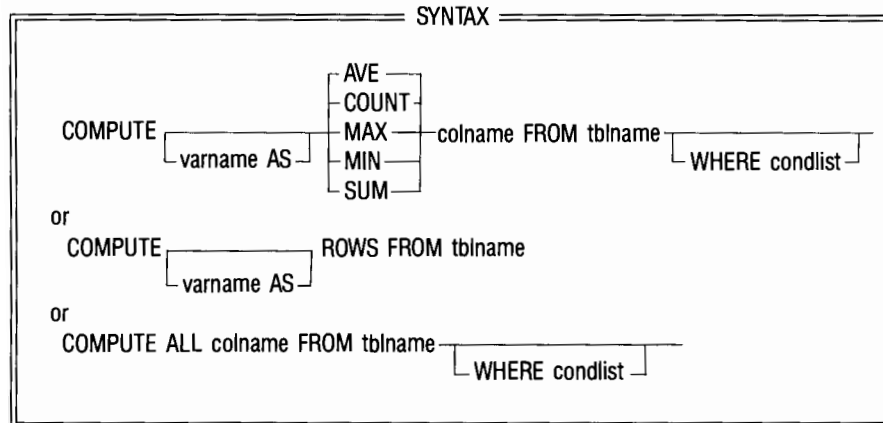
## PERFORMING ARITHMETIC CALCULATIONS

### Using the Compute Command

You can use the COMPUTE command to perform six arithmetic calculations on selected columns. The COMPUTE command can determine averages and totals for DOLLAR, INTEGER, and REAL columns. Minimums, maximums, and column and row counts can be computed on all data types.

COMPUTE is useful for values that only need to be calculated occasionally. For frequently calculated values, or for values that need to be used elsewhere in a table, it is best to store the arithmetic expression as a global variable or as a REPORT variable. For additional information on global variables see chapter 5. For additional information on REPORT variables see chapter 11.

The syntax for the COMPUTE command is:



You can use the COMPUTE command with a WHERE clause to limit the number of column rows that you want to compute. Additional information on the WHERE clause is in chapter 5.

The purpose for each calculation and all applicable restrictions are listed in table 8-2. Unless specified, the command is operational with all R:base data types.

Table 8-2 COMPUTE Command Arguments

Command	Function
ALL	Computes all six of the following mathematical operations.
AVE*	Computes the arithmetic average of DOLLAR, INTEGER, or REAL data types. Averages of integer values are rounded to the nearest integer value and dollars are rounded to pennies.
COUNT	Determines how many entries there are for a particular column item.
MAX*	Selects the row in a column with the greatest arithmetic or alphabetic value. For TEXT data types, the first 20 characters are evaluated.
MIN*	Selects the row in a column with the least arithmetic or alphabetic value. For TEXT data types, the first 20 characters are evaluated.
ROWS	Determines how many rows there are for a specific table.
SUM*	Computes the arithmetic sum of DOLLAR, INTEGER, or REAL data types.
*When these functions are performed on INTEGER or REAL vectors, then only the first value is used.	

When the COMPUTE command calculates averages, minimums, maximums, counts, and sums, null values are treated as zero. Averages that are calculated with rows that contain null values may yield results different from what you want. Null values can be excluded by using the WHERE clause (for example, WHERE *colname* EXISTS).

The following calculation results cause R:base to display a null value:

- INTEGER values greater than  $\pm 999,999,999$
- DOLLAR values greater than  $\pm 99,999,999,999,999.99$
- REAL values greater than  $\pm 9 \times 10^{\pm 37}$

As an example of how to use the COMPUTE command, suppose you want to know how many customers from Boston are listed in the table *custlist*. You need to count how many rows in the table have items in the *company* column and Boston in the *city* column. Your entry and the R:base response are similar to the following screen.

```
R> COMPUTE COUNT company FROM custlist WHERE city EQ Boston
company    Count      =      2
```

The *COMPUTE varname* option enables you to establish the result of the computation as a global variable. For example, the weekly sales for COMPUCO's Mary Simpson may be added to the running monthly total in order to update a sales report on a weekly basis. Rather than calculating the weekly total and writing it down for later use, you can assign the total a variable name. When you wish to update the monthly total, you can add the variable to the current monthly total.

The following command creates a variable named *wklysale* and calculates Mary Simpson's weekly sales total for a week in mid-March:

```
COMPUTE wklysale AS SUM price FROM transx WHERE empid = 160 AND +
tdate GE 03-11-85 AND tdate LE 03-15-85
```

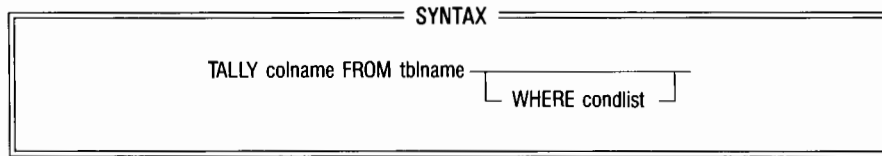
The result is stored as a variable. To see the value of the variable *wklysale*, use the SHOW VARIABLES command. Additional information on the SHOW command and variables is in chapter 5.

## TALLYING VALUES IN A COLUMN

### Using the Tally Command

The TALLY command counts the number of occurrences of each unique column value in a table. The information is sorted alphabetically and displayed. When you use the TALLY command to count TEXT type columns, only the first twenty characters of each entry are used to determine the uniqueness of the column. If a column contains many rows with unique values, R:base may not be able to tally every row in the table. If you enter the TALLY command and R:base displays an error message that tells you there are too many characters or values in the column, narrow the scope of the tally by adding a WHERE clause to the command.

The syntax for the TALLY command is:



The TALLY command can be used with a WHERE clause to limit the number of rows you want to look at. Additional information on the WHERE clause is in chapter 5.

For example, suppose you wish to list the computer models sold in March of 1985 by COMPUCO's Ernest Hernandez. You need to tally the number of occurrences of the product identification column (*prodid*) in the transaction table (*transx*) where the value in the employee identification column (*empid*) is Ernest's (#129) and the transaction date (*tdate*) is in the month of March 1985. Your command and the R:base response look like this:

```
R> TALLY prodid FROM transx WHERE empid = 129 and tdate eq 03-??-85
```

prodid	Number of Occurrences
CX3000	3
CX3010	1
CX3020	1
MB3000	2
MB3020	1
PB3040	2
PX3040	1



The results of a tally cannot be stored as a global variable. Similar results, however, can be achieved using the COMPUTE command. For each unique value, use the COMPUTE COUNT command with a WHERE clause and store the result as a variable.



---

## Data Modification Contents

<b>How to Use This Chapter</b>	9-2
<b>Assigning Calculated Values to a Column</b>	9-2
Using the ASSIGN Command	9-2
<b>Changing Column Values</b>	9-5
Using the CHANGE Command	9-6
<b>Deleting Rows from Tables</b>	9-7
Using the DELETE Command	9-7
<b>Editing Data Stored in Tables</b>	9-9
Using the EDIT Command	9-9
Editing with the Standard Display	9-9
Editing with the Forms Display	9-13



## HOW TO USE THIS CHAPTER

This chapter explains how to modify information stored in an existing database. Topics that are covered include removing rows from tables and modifying existing data. If you wish to change the structure of the database—for example, adding or deleting tables and columns—see chapter 6.

If you have defined passwords to protect your database, the data modification commands explained in this chapter may only be used by users with the correct user password. If you have defined rules, all modifications to the data are verified for compliance unless you use the `SET RULES OFF` command to turn off rule checking. See chapter 5 for further information on rules and chapter 6 for further information on passwords.

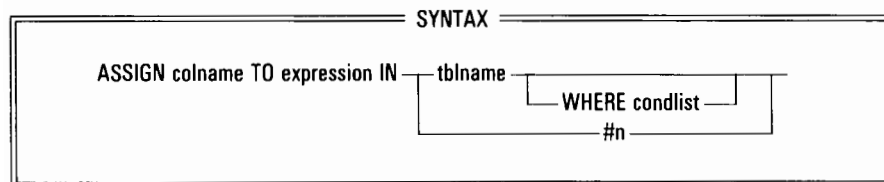
## ASSIGNING CALCULATED VALUES TO A COLUMN

### Using the ASSIGN Command

The `ASSIGN` command sets all values in a column, or values for selected rows, equal to the result of a concatenation or arithmetic expression. Expressions are made up of two items and an operator. The items are assigned numeric or text values and the operator determines what function is performed on the items.

If you have defined rules, all modified rows are verified for compliance unless you use the `SET RULES OFF` command to turn off rule checking. If a rule is violated, an error message is displayed and no modifications are made to the database. For further information on rules, see chapter 6.

The syntax for the `ASSIGN` command is:



The optional WHERE clause and row pointer (*#n*) option allow you to limit the rows that are assigned the value of the expression. Additional information on the WHERE clause is in chapter 5 and information on row pointers is in chapter 15.

The syntax for an expression must be one of the following:

```
colname op colname
colname op value
value    op colname
value    op value
```

The value of a global variable can be used instead of a numeric or text value. Precede the variable name with a period in the expression (for example, *.variable*). For additional information on global variables, see chapter 5. Table 9-1 lists the operators.

Table 9-1 Operators Used with the ASSIGN Command

Operator	Function	Example
+	Adds the two items, or combines two TEXT columns, variables, or values	12 + 13 = 25 jo + in = join
&	Combines two TEXT columns, variables, or values but leaves one space between	Sam & Evans = Sam Evans
-	Subtracts the second item from the first	25 - 10 = 15
/	Divides the first item by the second	16 / 2 = 8
X	Multiplies the two items	8 x 2 = 16
%	The first item is the percentage value of the second item*	50 % 16 = 8

\*Percentage is calculated as (item1 x item2) / 100. If both items are INTEGER, the result is INTEGER. If one or both items are REAL, the result is REAL.

The following guidelines apply to ASSIGN command expressions:

- Expressions with DOLLAR data types yield the following results:

DOLLAR + DOLLAR	=	DOLLAR	REAL	X	DOLLAR	=	DOLLAR
DOLLAR - DOLLAR	=	DOLLAR	REAL	%	DOLLAR	=	DOLLAR
DOLLAR / DOLLAR	=	REAL	INTEGER	X	DOLLAR	=	DOLLAR
DOLLAR X INTEGER	=	DOLLAR	INTEGER	%	DOLLAR	=	DOLLAR
DOLLAR X REAL	=	DOLLAR					
DOLLAR / REAL	=	DOLLAR					

- Expressions with DOLLAR and INTEGER, or DOLLAR and REAL data types are rounded to the nearest penny.

- Expressions with DATE data types yield the following results:

DATE - DATE	=	INTEGER (days)	DATE + INTEGER	=	DATE (new date)
DATE - INTEGER	=	DATE (new date)	INTEGER + DATE	=	DATE (new date)

- Expressions with TIME data types yield the following results:

TIME - TIME	=	INTEGER (seconds)	TIME + INTEGER	=	TIME (new time)
TIME - INTEGER	=	TIME (new time)	INTEGER + TIME	=	TIME (new time)

- When an expression is computed, values that exceed the limits of a column or variable are assigned null values (column limits are determined when the column is defined). If there is not enough room to display the value, an asterisk (\*) is displayed and a message indicates the number of values that are set to null.

- Any expressions that contain a null value produce a null result. R:base displays a message indicating how many values are set to null.

The ASSIGN command is useful for adjusting values in columns that require uniform changes. For example, suppose you want to increase the price of all COMPUCO products by five percent. You could use a calculator to determine each product's new price and use the edit command to change the *listprce* column, or you could apply the ASSIGN command.

Your entry for the command and the R:base response look like this on your screen:

```
R> ASSIGN listprc TO 105 % listprc IN product
The value for listprc has been changed in      27 row(s)
```

You can also use the ASSIGN command to combine two TEXT columns, variables, or values. Expressions with a plus sign (+) combine the two items with no space between them. Expressions with an ampersand (&) leave a space between the combined items. For example, suppose a table has a column for first names (*firstname*) and a column for last names (*lastname*) and both columns are 20 characters wide. One row contains the entries Sam and Jones. If you use an expression with a plus sign, such as

```
firstname + lastname
```

the result is:

```
SamJones
```

If you use an expression with an ampersand, such as

```
firstname & lastname
```

the result is:

```
Sam Jones
```

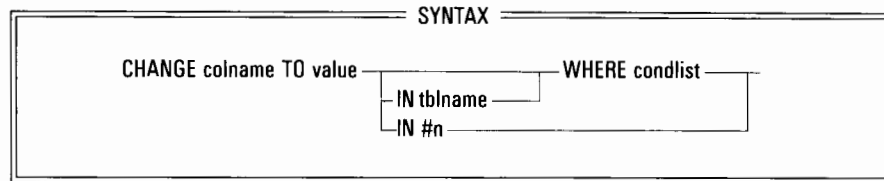
## CHANGING COLUMN VALUES

The CHANGE command changes one or more values of a column in a single table or in all tables in a database. The WHERE clause or row pointer option enables you to limit the rows that you want to change.

R:base also has a CHANGE COLUMN command that you can use to redefine the column structure. You can change the column name, data type, and length. For additional information on the CHANGE COLUMN command, see chapter 6.

## Using the CHANGE Command

The syntax for the CHANGE command is:



The value of a global variable can be used instead of a numeric or text value. Precede the variable name with a period (for example, *.variable*). For additional information on global variables, see chapter 5.

The optional *IN tblname* clause allows you to specify the table where the column is changed. If the *IN* clause is omitted, the column value is changed everywhere it exists in the open database.

The mandatory *WHERE* clause adds further qualifications that the data must meet, allowing you to precisely locate the rows that need modification. The optional *IN #n* expression allows you to use a row pointer instead of a *WHERE* clause to locate the row. The fundamental characteristics of the *WHERE* clause are shown in table 9-2. Additional information on the *WHERE* clause is in chapter 5 and information on row pointers is in chapter 15.

R:base assumes the value (or value of the global variable) is literal. Do not use the wildcard character (\*) as the value.

Table 9-2 Fundamental Characteristics of the *WHERE* Clause

WHERE:		
colname EQ value	colname1 EQA colname2	LIMIT EQ value
colname NE value	colname1 NEA colname2	COUNT EQ value
colname GT value	colname1 GTA colname2	COUNT EQ LAST
colname GE value	colname1 GEA colname2	colname EXISTS
colname LT value	colname1 LTA colname2	colname FAILS
colname LE value	colname1 LEA colname2	colname CONTAINS value

If you have defined data entry rules, all modifications to the data are verified for compliance unless you use the *SET RULES OFF* command to turn off rule checking. See chapter 5 for further information on rules and chapter 6 for further information on passwords.

In the following example, the change command is used to update a sales representative's name in the *compuco* database. Mary Simpson was married recently, and her new surname is Anderson. You could use the EDIT command to edit each table where Mary's name is stored, or you could use the CHANGE command to update the name in every table where it exists. The first name is stored separately from the last name, so define a WHERE condition that looks for verification of both first and last names.

Your entry for the command and the R:base response look like the screen below. Note that if there is only one Simpson surname in the database, it is not necessary to add the second half of the WHERE clause to verify the first name.

```
R>CHANGE lastname TO Anderson WHERE lastname = Simpson AND frstname = Mary
The value for lastname has been changed in      1 row(s)
```

## DELETING ROWS FROM TABLES

You can use the DELETE command to remove selected rows from tables or to remove duplicate rows from tables.

### Using the DELETE Command

Use the DELETE ROWS command to selectively remove rows of data from a table. After the command is executed, the space that was occupied by the deleted information is not available for use until it has been recovered. To reclaim the space, use the R:base PACK command explained in chapter 5.

The syntax for deleting selected rows from tables is:

SYNTAX	
DELETE ROWS FROM	tblname WHERE condlist
	#n

The mandatory WHERE clause or the optional row pointer (*#n*) expression allow you to pinpoint the rows you need to delete. Additional information on the WHERE clause is in chapter 5 and information on row pointers is in chapter 15.

In the following example, the company COMPUCO keeps a record of components used to make their products in a table called *compused*. All parts must have a valid product number and component number. You can use the DELETE command to remove any row that does not have an entry for either column. Your entry for the command and the R:base response look like this on the screen:

```
R>DELETE ROWS FROM compused WHERE prodid FAILS AND compid FAILS
1 row(s) have been deleted from compused
```

The DELETE command shown in the previous screen is also useful for removing forms and reports from a table. Additional information about the DELETE command and forms is in chapter 7. For additional information on the DELETE command and reports, see chapter 11.

Use the DELETE DUPLICATES command to remove duplicate rows from a table. Duplicate rows have identical column values for every column in the table. You are most likely to execute this command after you have performed some relational operations in order to create or update a table. The syntax for deleting duplicate rows from a table is:

```
DELETE DUPLICATES FROM tblname
```

When you execute the DELETE DUPLICATES command on large tables, processing time is decreased if at least one column in the table is defined as a key. For additional information on column keys, see chapter 6.

In the following example, duplicate rows are removed from the table *transx*. Your entry for the command and the R:base response are similar to the following:

```
R>DELETE DUPLICATES FROM transx
1 row(s) have been deleted from transx
```

## EDITING DATA STORED IN TABLES

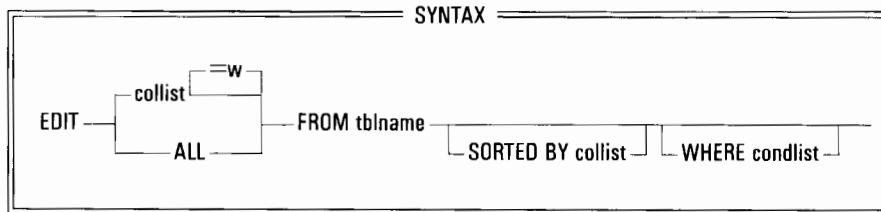
### Using the EDIT Command

You can use the EDIT command to modify some or all of the columns in a table, and to add or delete rows. There are two ways R:base can display the information in a database when the EDIT command is used. The standard display gives you access to an entire table and lets you browse through the rows displayed on the screen. The forms display lets you use a custom table form to review data one row at a time. If you have defined rules for a database or a table, either method tests all modifications for compliance unless you use the SET RULES OFF command to turn off rule checking. For further information on rules, see chapter 6.

### EDITING WITH THE STANDARD DISPLAY

The R:base standard display can be used with any table in the database. You can specify which columns you want to see, how wide each column display is, and how the rows are sorted.

When editing any or all columns in a table, the syntax for the EDIT command is:



The SORTED BY and WHERE clauses allow you to choose the order and range of the rows you want to edit. For additional information on the clauses, see chapter 5.



The width specification ( $=w$ ) enables you to adjust the column display width. For example, when you edit a table with ten integer columns, R:base normally displays seven complete columns and part of the eighth column. This is because the program allows ten spaces for each integer and one space between each column. If you know that the value in one column is never greater than 99 and that the value in another column is never greater than 9999, you can specify column widths of 2 and 4 for those particular columns. R:base takes advantage of the reduced column widths and displays more columns.

If you wish to adjust the display width of the column, enter an equals sign and the number of characters to be displayed immediately after the column name when you type the EDIT command. The default display width for TEXT columns is 40 characters and the maximum width is 80. When TEXT type data are larger than the defined display width, R:base continues the text on the next line in the column. With all other data types, when the column widths you select are not large enough to display the item on one line, asterisks are displayed to indicate an overflow condition.

After you enter the EDIT command, the table you select is displayed and a reverse video block, called the column cursor, highlights the item you are currently editing. If the table is larger than the screen, R:base indicates that there is more data to the left or right by displaying the word *more* and an arrow at the top of the screen. For example, if there are more columns on the right side of the table, the message *more*  $\rightarrow$  is displayed. You can move up, down, right, or left in the table. R:base allows you to scroll forward until you reach the end of the file, and to scroll backwards through the last fifty rows. Table 9-3 contains a complete list of the special function keys available with the standard display EDIT command.

When you edit data, R:base shows that you are changing information by displaying the word *update* at the top of the screen. After you modify the entry and move the cursor to another item in the table, the change is recorded and the update message is canceled. The changes that you make are not permanently recorded on the disk until editing is complete and you press the [ESC] key.

Table 9-3 Special Function Keys Used with the Standard Display EDIT Command

Key(s)	Function
[Tab]	Moves the column cursor right one column
[Shift] [Tab]	Moves the column cursor left one column
[→]	Moves the cursor to the right within the highlighted area
[←]	Moves the cursor to the left within the highlighted area
[↑]	Moves the column cursor up one row
[↓]	Moves the column cursor down one row
[Ctrl] [→]	Moves the column cursor to the right-most column in the row
[Ctrl] [←]	Moves the column cursor to the left-most column in the row
[Home]	Moves the column cursor to the upper left column in the rows that are displayed
[End]	Moves the column cursor to the lower right column in the rows that are displayed
[PgUp]	Shows the previous page of data
[PgDn]	Shows the next page of data
[Del]	Removes the character at the current cursor position
[Ins]	Inserts a space at the current cursor position
[F2]	Deletes the row at the current cursor position
[F5]	Resets the column entry to its original value. Not operational after the column cursor is moved



The following screen shows the result of an EDIT command that displays every column and row in a table. The table *product*, an inventory file, is made up of seven columns and 27 rows. The command:

EDIT ALL FROM product

results in a screen similar to the following:

Press [ESC] to quit, [F2] to delete, [F5] to reset			more→
prodid	proddesc	prodname	onh
CX3000	System-256K-Dual Drive	Standard Color PC	
CX3010	System-512K-Dual Drive	Classic Color PC	
CX3020	System-Dual Drive w/Hard Disk	Advanced Color PC	
CX3030	System-Single Drive w/Hard Disk	Superior Color PC	
MB3000	System-512K-Dual Drive	Standard PC	
MB3020	System-Dual Drive w/Hard Disk	Advanced PC	
MB3030	System-Single Drive w/Hard Disk	Superior PC	
MX3000	System-256K-Dual Drive	Standard PC w/ deluxe keyboard	
MX3010	System-512K-Dual Drive	Classic PC w/ deluxe keyboard	
MX3020	System-Dual Drive w/Hard Disk	Advanced PC w/ deluxe keyboard	
MX3030	System-Single Drive w/Hard Disk	Superior PC w/ deluxe keyboard	
PB3040	System-256K-Dual Drive-Portable	Portable Standard PC	
PB3050	System-512K-Dual Drive	Classic Color PC	
PX3040	System-256K-Dual Drive-Portable	Portable Standard PC w/ deluxe kbd	
PX3050	System-512K-Dual Drive	Classic Color PC w/ deluxe kbd	
1000	Basic Keyboard	standard keyboard component	
1010	Deluxe Keyboard	deluxe keyboard component	

The first three columns of *parts* are completely displayed, and the fourth column, *onhand*, is partially displayed. Suppose you want to:

- Display all columns on the screen at the same time
- List the product description first
- Sort the rows by the product number

Enter the command

```
EDIT proddesc=10 prodid=6 prodname=21 onhand=5 cost=7 +
listprc=7 location=4 FROM product SORTED BY prodid
```

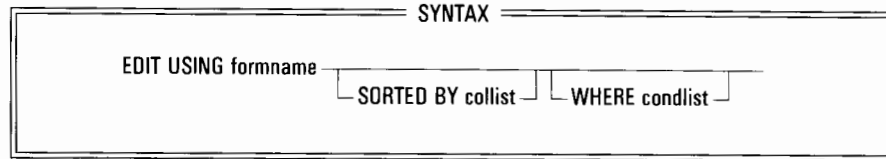
and a screen similar to the following appears.

Press [ESC] to quit, [F2] to delete, [F5] to reset						
proddesc	prodid	prodname	onhan	cost	listprc	loca
Basic Keyboard	1000	standard keyboard component	20	250.00	315.00	A-50
Deluxe Keyboard	1010	deluxe keyboard component	25	280.00	346.50	A-60
Basic Keyboard-P	1020	basic portable keyboard component	200	300.00	367.50	A-10
Deluxe Keyboard-P	1030	deluxe portable keyboard component	175	350.00	420.00	A-30
Monitor-Color-12	2000	standard monitor	200	150.00	210.00	B-10
Monitor-Monochrome-12	2010	color monitor component	225	250.00	315.00	B-20
Box-256K-Dual Drive	3000	256K dual drive component	200	1000.00	1575.00	C-10
Box-512K-Dual Drive	3010	512K dual drive component	150	1200.00	1785.00	C-20
Box-Dual	3020	512K dual drive	75	1600.00	2205.00	C-30

### EDITING WITH THE FORMS DISPLAY

The R:base forms display can be used only with tables that have a defined entry form. As with the standard display, you can select the sorting order of the rows. Unlike the standard display, when the EDIT command is entered and a form name is specified, the data in the table is displayed one row at a time using the format that you defined when you made the form. For more information on forms, see chapter 7.

When R:base forms are used to edit data, each row in the table is displayed only once. To edit data using a form, the syntax for the EDIT command is:



When the EDIT command is used with a form, the menu shown in the following screen appears above the first row of information in the table. The menu options are described in table 9-4.

--Skip--Edit--Change--Add--Reset--Delete--Quit-----

Table 9-4 Menu Options for the R:base Forms Display EDIT Command

Option	Result
Skip	The row displayed on the screen is not modified and the next row in the table is displayed.
Edit	The row displayed on the screen may be changed. Press [E] and modify the row. When you are done, press the [ESC] key to return to the EDIT command menu and choose one of the four following options: <i>Change</i> , <i>Add</i> , <i>Reset</i> , or <i>Delete</i> .
Change	The modified row on the screen is saved and the next row in the table is displayed.
Add	The row displayed on the screen is added as a new row to the table and the original row is left unchanged. You now have two rows of data. The original row retains the same place in the table, and the added row is the last row in the table.
Reset	The row displayed on the screen is not saved. R:base ignores the modifications you made to the row and resets the row to its original value. If the <i>Change</i> or <i>Add</i> options have already been entered, <i>Reset</i> will not recall the original values.
Delete	The row displayed on the screen is deleted from the database when you confirm the command and the next row in the table is displayed.
Quit	This option terminates the EDIT mode. All modifications are permanent after <i>Quit</i> is executed. R:base displays the number of rows that were changed, deleted, or added and returns to the R > prompt.

Choose the menu option by entering the first letter of the command or by moving the cursor to the option and pressing [ENTER].

The following screen shows a typical form that has information which needs editing. After selecting the *Edit* option, move the column cursor to the items you wish to modify. When all changes to the displayed row are complete, press [ESC] to return to the main menu. Then choose the appropriate option to complete the editing process.

```
Press [ESC] when done with this data
Transaction number (Be sure to clear this with Ruth!) 4792
Your employee number (Pete-133 Ernie-129 Mary-160 John-131) 129
Customer ID number (Listed in table called custlist) 102
What did you sell? (List the product number) CX3020
How many units did you sell? 5
What was the price per unit? 2,575.00
What is the transaction date? 12-13-84
```

After every row has been displayed once, or after the *Quit* option is selected, a message informing you how many rows have been changed, deleted, and added appears.



---

# Relational Operations Contents

<b>How to Use This Chapter</b>	10-2
<b>Relational Operations</b>	10-2
<b>Project</b>	10-3
<b>Append</b>	10-5
<b>Union</b>	10-7
<b>Intersect</b>	10-9
<b>Subtract</b>	10-11
<b>Join</b>	10-13





## HOW TO USE THIS CHAPTER

This chapter explains the purpose of the six relational commands in R:base and the requirements and options for entering each. It also gives you examples and suggestions for using the commands. Before you use the commands in this chapter, you need to have defined your database and, in all but one case, your database must contain at least two tables. You should also be familiar with the SELECT command.

Read the section on “Relational Operations” first. Then, if you are an experienced user of R:base, go on to any of the commands described in this chapter. The discussion of each command is sufficiently independent so you need only read the sections for which you have immediate use. If you are a new user, you will find it helpful to read through the entire chapter.

## RELATIONAL OPERATIONS

The R:base relational commands provide a formal set of tools you can use to manipulate the structure of the tables and the data in your database. For example, they allow you to store your data in small tables that you can combine later for reports and particular needs as they arise. They eliminate any need for redundant entries by allowing you to move data from existing tables to a new table.

R:base contains six relational commands. Use the PROJECT command to make a new table that is a complete or partial copy of an existing table. Use APPEND to take rows from one table and add them to the rows of another table. Use UNION, INTERSECT, SUBTRACT, and JOIN to make a new table from two that exist. With PROJECT, you have the option of sorting. With PROJECT and APPEND, you can specify a WHERE condition. With PROJECT, UNION, INTERSECT, and SUBTRACT, you can choose the columns you want included in the new table. These options give you control over the structure of the new table you create, the selection of data from the existing tables, and its arrangement in the new table.

With the exception of the APPEND command, these commands copy data to new tables. The sending or source tables are not changed.

You can increase the speed of the UNION, INTERSECT, SUBTRACT, and JOIN commands by making the common column keyed in the second table (*tblname2* in the syntax for the command). These commands do not use keys in the first table (*tblname1*). If one table has many more rows than the other, you can increase the speed of the UNION, INTERSECT, and JOIN commands by making the small table second in the command and by not making it keyed. If both tables have many rows, key the common column in the second table and place the table with the shorter row length first.

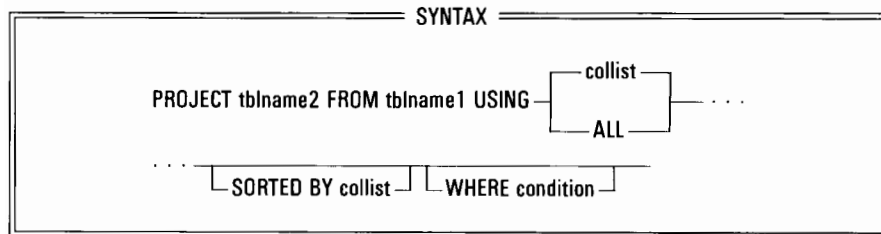
## PROJECT

The PROJECT command creates a new table from an existing table. You can specify one or more of the columns from the existing table and their order. You can also specify the rows to transfer with a WHERE clause and have them sorted. The command is useful for purposes such as:

- Making a small table from a large one when you need only part of the large one
- Making a back-up copy of a table within the database
- Rearranging the order of columns in the row
- Rearranging the order of rows in the table

R:base copies the columns and rows that you specify to the new table. In the new table no columns are keyed. If you have any keyed column in the source table that you want keyed in the new table, follow the PROJECT command with the BUILD KEY command to create the keys.

The syntax of the PROJECT command is:



You must include the USING clause. If you are selecting a subset of columns from the existing table, name them. If you are selecting all columns, use the word ALL in the clause. The WHERE clause and the SORTED BY clause are optional.

Figure 10-1 shows the operation of the PROJECT command. The columns *lastname* and *empid* are selected from the existing table *emptable* for the new table *newtable*. The order of the columns in the USING clause puts *empid* before (to the left of) *lastname*. The SORTED BY clause places the rows in ascending numerical order by employee number. Finally, the WHERE clause selects only rows where the employee hire date is before January 1, 1985.

Your entry for this example and the R:base response look like this on your screen:

```
R>PROJECT newtable FROM emptable USING empid lastname SORTED BY empid +  
R>WHERE hiredate LT 01/01/85  
Successful project operation          3 rows generated
```

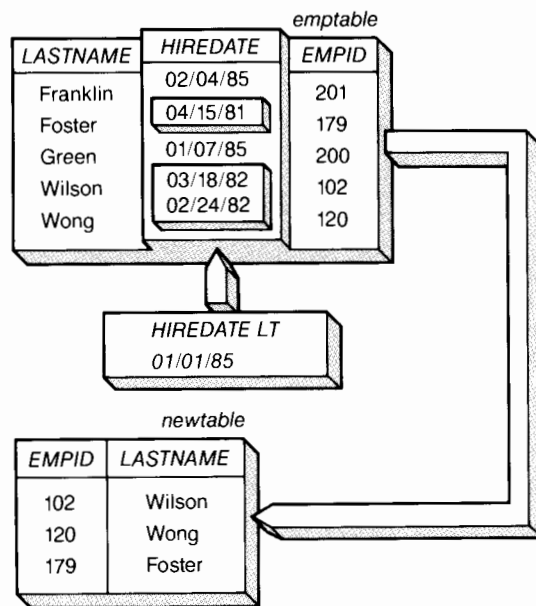


Figure 10-1 Using the PROJECT Command to Form *Newtable*

## APPEND

The APPEND command copies rows from one table to the end of another table. It can also copy rows from the same table to itself. Unlike the other relational commands, APPEND does not create a new table. Instead, R:base lengthens the destination table with the rows from the source table. The source table is not affected by the command.

R:base appends data only for columns that exist in both tables. For columns in the destination table that are not in the source table, R:base provides null values in the appended rows.

You can use a WHERE clause with this command to specify conditions for the selection of rows that R:base will append.

You can use the command to append a day's worth of entry data onto a table that holds month-to-date data. You can enter daily transactions into a temporary work table, make certain that your entries are correct, append the temporary table to the month-to-date table, and then delete all the rows from the daily table.

The syntax of the APPEND command is:

SYNTAX	
APPEND tblname1 TO tblname2	WHERE condition

The first table (*tblname1*) holds the rows to be appended. The columns in the second table (*tblname2*) need not be in the same order as in the first table.

Figure 10-2 shows the operation of the APPEND command. *Newemp*, a table with data about new employees, is appended to *emptable*, the table with data about old employees. R:base adds rows containing data for *lastname*, *hiredate*, and *empid*, since the columns exist in both tables. The column *lastcomp* in *newemp*, however, does not exist in *emptable*, so the column and its data are not appended. On the other hand, the column in *emptable* that holds telephone extension numbers is not in *newemp*. Since there is no data to be appended, R:base fills the new rows with nulls for that column.

Your entry and the response from R:base look like this on your screen:

```
R> APPEND newemp TO emptable
Successful append operation    2 rows added
```

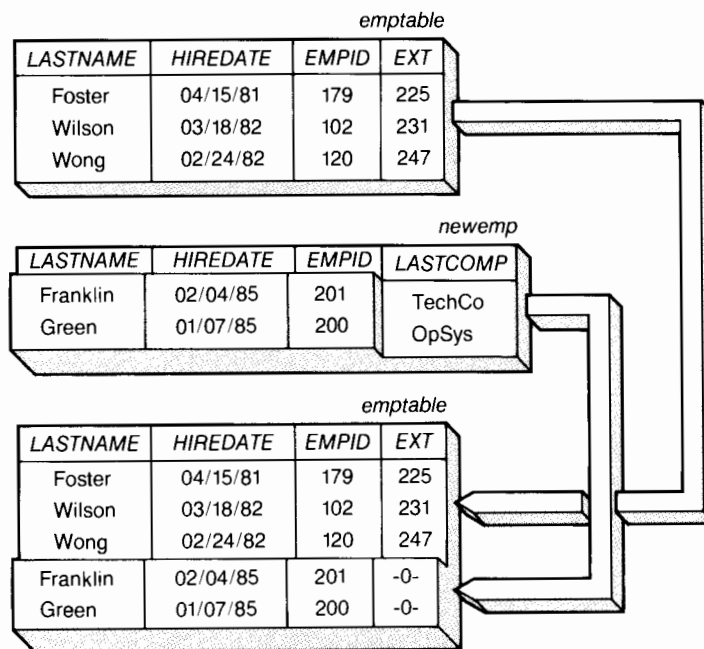


Figure 10-2 Using the APPEND Command

## UNION

The UNION command combines the columns and rows of two existing tables in a new table. R:base looks for columns that have the same name in both tables. If R:base finds at least one common column, it builds a new table that contains all columns from the two tables, combining each of the two common columns into one.

If the same values for the common columns are found in each table, R:base combines the two rows into one in the new table. For rows with no matching values, nulls are used to fill out the remaining column values. For example, if two tables have the column *lastname* in common, R:base can combine the tables. If a row from each has the value *Smith* in the *lastname* column, R:base combines the two rows into one. If a row from one table has the value *Smith* and a row from the other table has the value *Jones*, R:base copies both rows to the new table, adding to each row the columns from the other row, filled with nulls.

If a row has two or more columns in common with a row from the other table, all pairs of values from common columns must match for the pairs of rows to be combined.

This is the syntax for the command:

SYNTAX	
UNION tblname1 WITH tblname2 FORMING tblname3	<div style="border: 1px solid black; display: inline-block; padding: 2px 10px;">             USING collist           </div>

If you do not include a USING clause, the first table (*tblname1*) provides the order for rows and columns in the new table. Columns from the second table (*tblname2*) that are not in the first are added to the right of the columns from the first table.

The optional USING clause lets you choose one or more columns (*collist*) from either one of the existing tables or from both. Also you can use the clause to determine the order of the columns in the new table. At least one of the columns in your USING clause must exist in both tables. If you do not include the clause, R:base uses all columns.

Figure 10-3 shows the operation of the UNION command. *Emptable* is united with *pay84*. The common column *empid* is combined in the new table. R:base provides nulls for column values when it finds no matching values, as in the case of *pay* in two rows of *emptable* and of *lastname* and *hiredate* in one row of *pay84*.

Your entry for this example and the response from R:base look like this on the screen:

```
R> UNION emptable WITH pay84 FORMING newtable
Successful union operation      6 rows generated
```

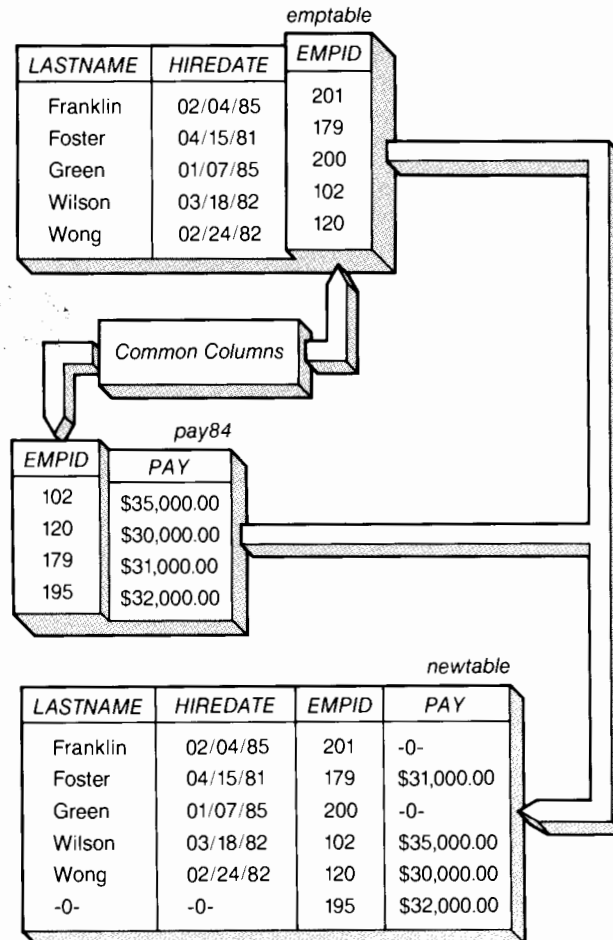


Figure 10-3 Using the UNION Command to Form *Newtable*

## INTERSECT

The INTERSECT command combines the columns and rows of two existing tables in a new table if R:base finds at least one common column. Each set of values from common columns must match in a row from the first table and a row from the second table for the rows to be included in the new table. For example, if each table has the column *lastname* in common, R:base will intersect a row from each if the column in each row has the same value, such as *Smith*. R:base looks in each table for at least one column that has the same name. It compares each row of the first table with each row of the second table. When it finds a column with a value that matches that of the other table, it combines the two rows into one.

If a row from one table has more than one column in common with a row from the other table, all pairs of column values have to match. If R:base does not find a match between every pair of column values in the two rows, it does not include a combined row in the new table. For example, if both rows have the columns *firstname* and *lastname*, R:base will intersect a row from each if *John* is the first name in both first-name columns and *Smith* is the name in both last-name columns. It will not, however, intersect the rows if *Smith* is the last name in both but *Mary* is the first name in one and *John* in the other.

This is the syntax for the command:

Computer  
Museum

```

SYNTAX
INTERSECT tblname1 WITH tblname2 FORMING tblname3
        USING collist
  
```

If you do not include a USING clause, the first table (*tblname1*) provides the order for rows and columns in the new table. Columns from the second table (*tblname2*) that are not in the first are added to the right of the columns from the first table.

The optional USING clause lets you choose one or more columns (*collist*) for the new table. Also you can use the clause to arrange the order of the columns in the new table. At least one column that you list in the clause must exist in both tables.

If your tables have more than one column in common, R:base must find a match between all sets of values. If you have a USING clause, the match is limited to just the common columns you include. If you do not have the clause, R:base checks all columns and places matching values in the new table.

Figure 10-4 shows the operation of the INTERSECT command. *Emptable* is intersected with *pay84*. R:base, finding a match for three rows from each table, copies them to *newtable*. Two employee numbers in *emptable*—200 for Green and 201 for Franklin—do not match employee numbers in *pay84*. R:base does not copy those rows to *newtable*. One employee number in *pay84*—160—does not match employee numbers in *emptable*. R:base does not copy the row that includes the non-matching column.



Your entry for the example and the response from R:base look like this on the screen:

```
R> INTERSECT emptable WITH pay84 FORMING newtable
Successful intersect operation      3 rows generated
```

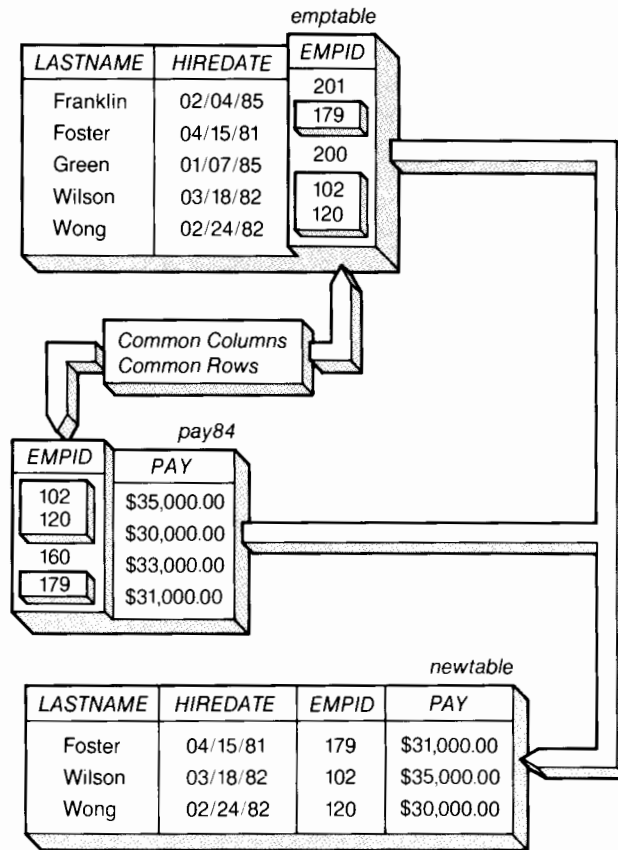


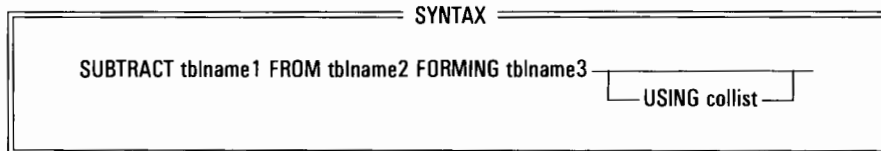
Figure 10-4 Using the INTERSECT Command to Form *Newtable*

## SUBTRACT

From two existing tables, the SUBTRACT command forms a new table made up of rows from one table that do not match those of the other. R:base looks in each table for columns that have the same name. R:base compares each row of the first table with every row of the second table. Each row of the second table that has no match in the first table is copied to the new table.

This command is the opposite of the INTERSECT command. If you use commands with the same two tables, you will see that SUBTRACT finds a different set of rows from those INTERSECT finds. The UNION command, on the other hand, finds all rows.

This is the syntax for the command:



The first table (*tblname1*) that you enter in the command is subtracted from the second. The phrasing of the command is similar to that of an arithmetic subtraction, such as "subtract 10 from 100 to get (form) 90." If you do not include a USING clause, the new table (*tblname3*) uses the rows from and follows the column order of the second table (*tblname2*).

The optional USING clause lets you choose one or more columns (*collist*) for the new table from the second existing table. You can also use the clause to arrange the order of the columns in the new table. At least one column that you list in the clause must exist in both tables. If you do not include the clause, R:base checks all columns and places the rows from the second table into the new table when there is no match.

Figure 10-5 shows the operation of the SUBTRACT command. The tables *pay84* and *emptable* both have the column *empid*. *Pay84* contains the 1984 salary for employees. Subtracting *pay84* from *emptable*, the list of current employees, will show which employees received no pay in 1984. R:base, finding that new employees Franklin and Green do not have employee numbers in *pay84*, copies their rows from *emptable* to the new table.

The entry for this example and the response from R:base look like this on your screen:

```
R> SUBTRACT pay84 FROM emptable FORMING newtable
Successful subtract operation      2 rows generated
```

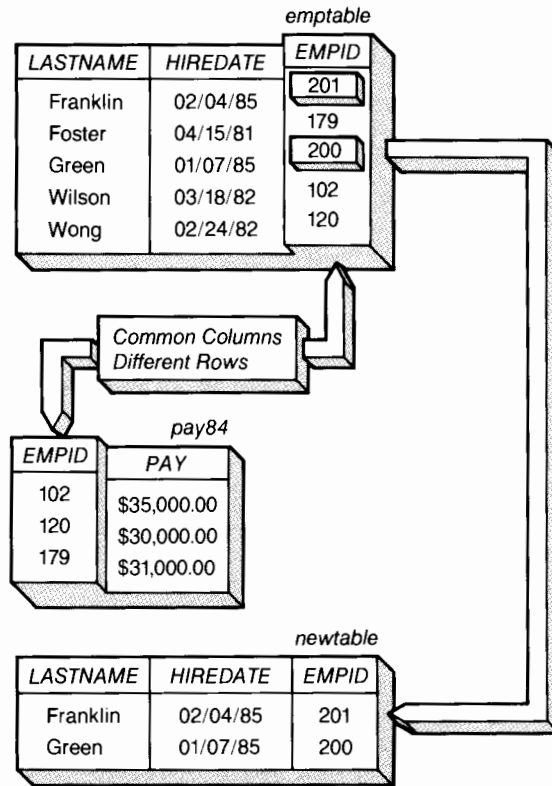


Figure 10-5 Using the SUBTRACT Command to Form *Newtable*

## JOIN

Like the UNION command, the JOIN command forms a new table by combining rows from two existing tables. The JOIN command, however, performs two operations the UNION command does not do:

- R:base *compares* a column from the first table with a column from the second table. You choose one of six conditions for the comparison. For example, the value for the column in a row from one table is *equal to* the value for the column from the second table—*Smith = Smith*; or the value from table one is *greater than* that from the second table—*\$2,000 > \$1,500*. R:base copies to the new table each row that satisfies your specified comparison.
- The two columns you choose for comparison have different names. They must, however, be of the same data type and length. For example, you can compare columns named *lastyear* and *thisyear* if both are DATE data types but not if one is TEXT and the other DATE. The two columns used for the comparison as well as all other columns from both tables will appear in the new table.

The syntax for this command is:

SYNTAX	
<pre> JOIN tblname1 USING colname 1 WITH tblname2 USING colname 2 ... FORMING tblname3 WHERE logicalop           </pre>	<pre> ----- ----- -----           </pre>

R:base places the columns from the first table (*tblname1*) first in the new table—on the left. Columns from the second table (*tblname2*) are placed second in the new table—on the right. Unlike the UNION command, the JOIN command does not combine columns that exist in both tables. Values for each appear separately in the row.

Although R:base allows you to use the same column name for both *colname1* and *colname2*, it is recommended for best results that you either rename one of the columns before you use the JOIN command, rename one of the columns before using the new table, or use the UNION command rather than the JOIN command.

For setting up the comparison, you use a unique kind of WHERE clause. You may omit the WHERE clause if you choose the logical operator *equals*, since this is the default. To specify any comparison other than *equals*, use a WHERE clause. See table 10-1.

Table 10-1 Logical Operators for the JOIN Command WHERE Clause

Operator	Rows are included if
EQ	column1 equals column2
NE	column1 is not equal to column2
GT	column1 is greater than column2
GE	column1 is greater than or equal to column2
LT	column1 is less than column2
LE	column1 is less than or equal to column2

Making column two (*colname2*) keyed, as described above in “Relational Operations,” increases the speed of processing only if the comparison is *equals*.

Figure 10-6 shows the operation of the JOIN command. The column *lastname* from the employee table *emptable* is compared with *manager* from the table *mgrtable*. That is, R:base looks for identical data in the two columns it compares. In the new table, Wong is listed on two rows, since he manages two departments and, therefore, has two matches for the comparison. Franklin and Green, who are not managers, are excluded, since they have no matching column in *mgrtable*.

Your entry for this example and the response from R:base look like this on your screen:

```
R> JOIN emptable USING lastname WITH mgrtable USING manager
Successful join operation      4 rows generated
```

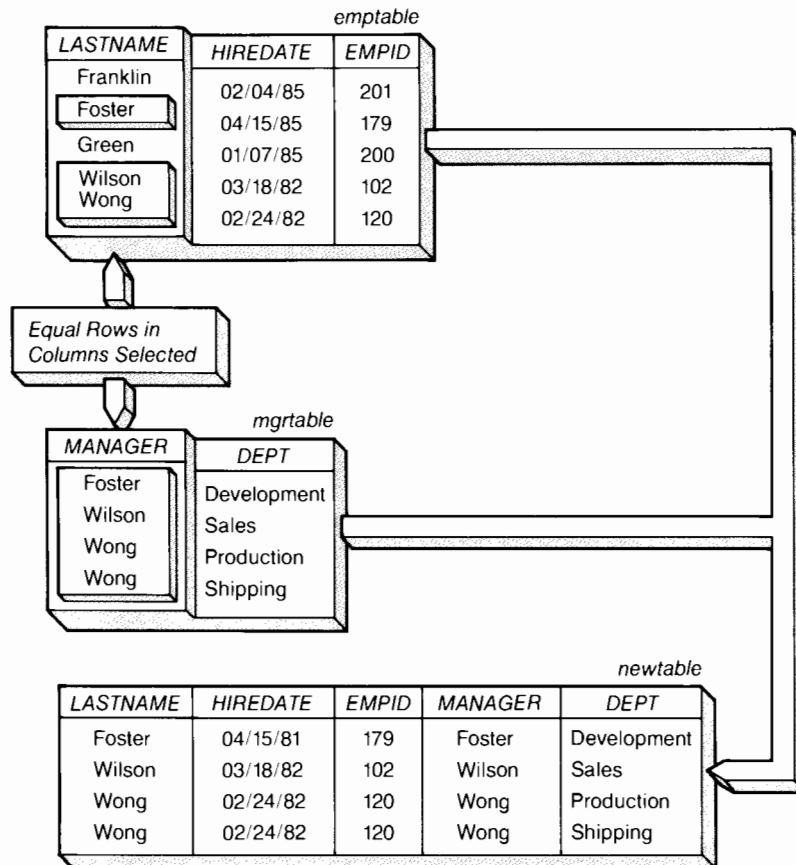
Figure 10-6 Using the JOIN Command with the Default *Equals*

Figure 10-7 shows another example of the JOIN operation. In this case, the WHERE clause is included to specify that the comparison is *greater than*. The first table, *repsales*, lists the accumulated sales for the sales representatives. The second table, *contest*, lists the awards that will be given to the sales representatives during a sales contest. To be eligible for any award listed, a representative's sales must exceed the total shown in the column *amount*. The JOIN command creates a new table, *winners*, that will show the results of the contest. In the command, the columns *sales* and *amount* are specified. The WHERE comparison states the figure for *sales* must be greater than that for *amount*.

Your entry for the example in figure 10-7 and the response from R:base look like this on your screen:

```
R> JOIN repsales USING sales WITH contest USING amount WHERE gt
Successful join operation      6 rows generated
```

Notice that for each set of figures that satisfies the comparison, R:base places a new row in the new table, repeating in each of these rows the values for *lastname* and *sales*. The row for Coffin is not joined and copied, since his sales figure is not greater than any contest amount.

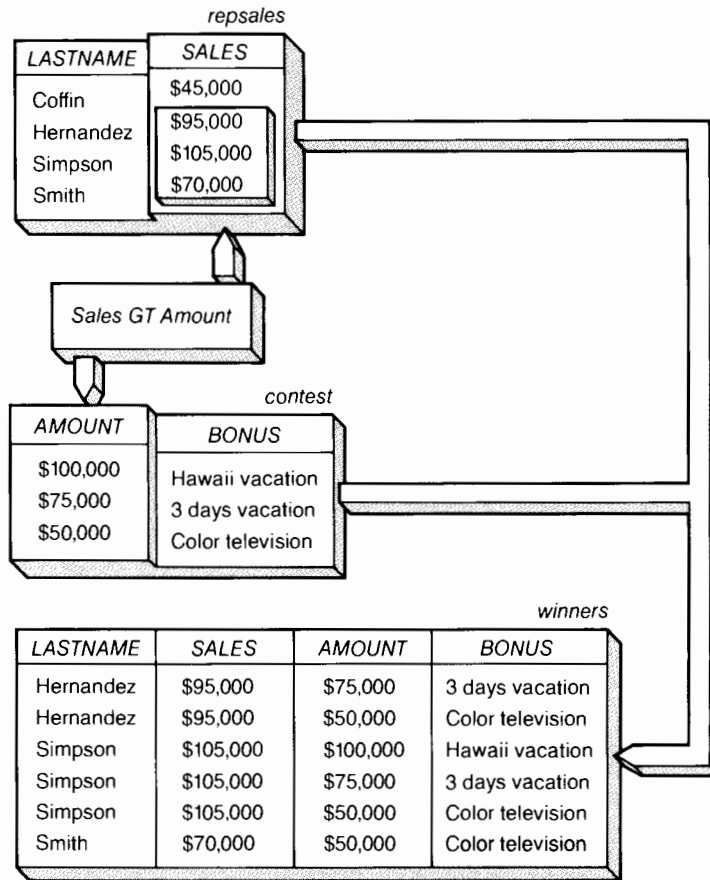


Figure 10-7 Operation of the JOIN Command with *Greater Than*





## Reports Contents



<b>How to Use This Chapter</b>	11-3
<b>R:base REPORTS Overview</b>	11-4
REPORTS Features and Capabilities	11-4
The Sample Report	11-4
Report Design	11-7
<b>Creating the Report</b>	11-7
Choosing a Report Name	11-8
Help for Report Generation	11-10
<b>Editing the Report</b>	11-11
Editing the Report Format	11-11
The Report Sections	11-11
Keyboard Functions	11-12
Defining, Changing, and Deleting Report Variables	11-14
Defining New Variables	11-14
Calculating Report Subtotals	11-16
Calculating Report Totals	11-17
Calculating Percentages	11-17
Defining Columns from Other Tables	11-18
Defining Date, Time, and Page Numbers	11-20
Combining Text Expressions	11-21
Using Constants in Expressions	11-21
Using Global Variables in Expressions	11-22
Setting Printer Control Variables	11-22
Changing Existing Variables	11-23
Reordering, Deleting, and Data Typing Variables	11-23
Reordering Variables	11-23
Deleting Variables	11-23
Changing Variable Data Types	11-23
Example Report Variables	11-24
Locating Table Columns and Variables on the Report	11-24
Locating New Variables and Table Columns	11-25
Setting Text Wrap Locations	11-26
Masks for DOLLAR Data Types	11-29
Changing and Deleting Locations of Variables and Table Columns	11-29

✱ Marking Heading, Detail, Breakpoint, and Footing Lines	11-30
✱ Report Heading and Footing Option	11-32
✱ Page Heading and Footing Option	11-35
✱ Detail Option	11-38
✱ Breakpoint Subheadings and Subfootings Option	11-40
✱ Setting Page Length	11-44
✱ Leaving Report Generation	11-44
<b>Printing the Report</b>	11-45
✱ Output to the Computer Screen	11-46
✱ Output to the Printer	11-46
✱ Printing on Single Sheets	11-47
✱ Output to a File	11-47
<b>Deleting Reports</b>	11-48
<b>Advanced Report Concepts</b>	11-48
✱ Using Variables and Route Numbers	11-48
✱ Making Mailing Labels	11-49
✱ Creating Form Letters	11-51
✱ Duplicating a Report Format	11-53
✱ Duplicating a Report Within a Database	11-53
✱ Copying a Report to a Different Database	11-54
✱ Decimal Point Alignment for Real Numbers	11-55

## HOW TO USE THIS CHAPTER

This chapter contains information on each phase of report formatting and production. Since REPORTS is a menu-oriented subsystem within R:base, the flow of this chapter is based on the flow of these menus. Figure 11-1 illustrates the menus of the REPORTS subsystem.

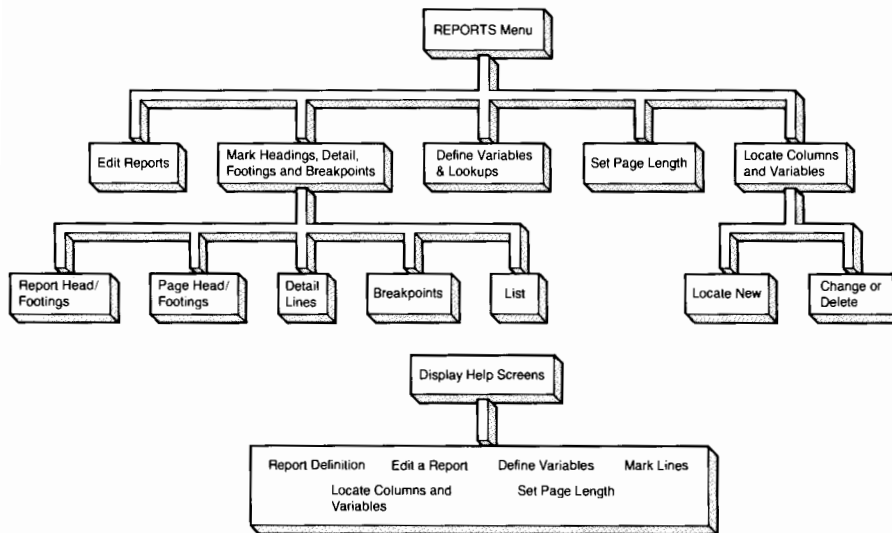


Figure 11-1 REPORTS Menus

## **R:BASE REPORTS OVERVIEW**

### **REPORTS Features and Capabilities**

With the R:base REPORTS option you can create a variety of reports. You can design form letters, sales and expense reports, and statements. You can print information on preprinted forms, such as checks, invoice forms, and insurance forms. For input to other programs, you can also create specially formatted files.

The full-screen editing functions of the REPORTS option allow you to create reports to your own specifications, drawing data from one or more tables. You can compose a report for any R:base table and specify text for a heading, detail lines, a footing, and up to 10 levels of subfootings and subheadings. Information for your report can be taken directly from table columns or from defined variables.

You can use the R:base OUTPUT command to display your report on your screen, send it to your printer, or send it to a disk file.

REPORTS features allow you to:

- Use a large report format: up to 131 columns wide with up to 66 line definitions (up to 999 lines per printed page)
- Define up to 40 variables in your reports
- Define up to ten levels of summary breakpoints — subheadings and subfootings
- View cursor screen line and column values while editing
- Retrieve and print information from any column in any table
- Paragraph long textual information in a specified format
- Incorporate the current time, date, and page number in your report
- Combine string variables
- Pause at the end of the page to change forms
- Send printer control codes for special effects such as compressed print
- Format dollar amounts for check processing

### **The Sample Report**

In this chapter, the Daily Sales Transaction Report illustrated in figure 11-2 is the basis for examples and explanations on how to create reports in R:base. The sample report contains information from the *compuco* database.

The *transx* table contains most of the information printed on the report. Columns from other tables must be defined as *lookup* variables (see “Defining Columns from Other Tables”). Table 11-1 lists all table columns used on the report, the table they are from, and the report heading under which they appear.

Table 11-1 Sample Report Tables and Columns

Table	Column	Contents	Report Heading
transx	transid	Transaction Number	Receipt#
	custid	Customer Identification Number	Cust#
	empid	Employee Identification Number	Sls#
	prodid	Product Number	Product#
	unit	Number of units sold	Units
	price	Price per unit sold	Price
custlist	company	Customer company name*	Customer Name
salesrep	firstname	Sales representative's first name*	Salesman Name
	lastname	Sales representative's last name*	Salesman Name
product	cost	Product unit cost*	Cost

\*Must be defined as lookup variables.

Some of the information appearing on the report results from a computation on data held in one or more of the tables. Computations are accomplished by defining variable expressions. Table 11-2 lists the variables defined for the sample report.

Table 11-2 Sample Report Variable Computations

Variable Name	How Calculated	Heading on Report
extprice	Product price times units	Ext. Price
extcost	Product cost times units	Ext. Cost
subunits	Subtotal of units sold per customer	-----
subprice	Subtotal of extended prices for all products per customer	-----
subcost	Subtotal of extended costs for all products per customer	-----
totunits	Total of all units sold	=====
totprice	Total of all extended prices	=====
totcost	Total of all extended costs	=====
profit	Profit percentage derived from total extended price, total extended cost, and total units sold	Profit:

COMPATIBLE COMPUTER COMPANY DAILY SALES TRANSACTIONS							
							Date 03/11/85
							Page 1
Cust#	Customer Name	Sls#	Salesman Name	Units	Unit Price	Ext. Price	Cost
	Receipt#	Product#					Ext. Cost
100	PC Distribution Inc.	129	Ernest Hernandez				
	4790	CX3000	15	\$1,800.00	\$27,000.00	\$1,530.00	\$22,950.00
	4796	PB3050	8	\$3,000.00	\$24,000.00	\$2,300.00	\$18,400.00
	5081	PX3050	5	\$3,000.00	\$15,000.00	\$2,350.00	\$11,750.00
	5060	MX3010	10	\$1,700.00	\$17,000.00	\$1,630.00	\$16,300.00
	5080	PB3040	15	\$2,250.00	\$33,750.00	\$2,175.00	\$32,625.00
	Customer 100	Subtotal	53		\$116,750.00		\$102,025.00
101	Computer Distributors	102	June Wilson				
	4875	MX3000	50	\$1,600.00	\$80,000.00	\$1,430.00	\$71,500.00
	4794	CX3030	50	\$2,875.00	\$143,750.00	\$2,130.00	\$106,500.00
	4795	MB3000	10	\$1,800.00	\$18,000.00	\$1,400.00	\$14,000.00
	Customer 101	Subtotal	110		\$241,750.00		\$192,000.00
102	Industrial Computers	131	John Smith				
	5075	PB3040	30	\$2,150.00	\$64,500.00	\$2,175.00	\$65,250.00
	5055	CX3000	50	\$1,750.00	\$87,500.00	\$1,530.00	\$76,500.00
	Customer 102	Subtotal	80		\$152,000.00		\$141,750.00
103	Computer Mountain Inc.	131	John Smith				
	4970	MX3010	35	\$1,875.00	\$65,625.00	\$1,630.00	\$57,050.00
	5000	PB3040	35	\$2,500.00	\$87,500.00	\$2,175.00	\$76,125.00
	Customer 103	Subtotal	70		\$153,125.00		\$133,175.00
104	Industrial Concepts	131	John Smith				
	5005	PB3050	12	\$3,175.00	\$38,100.00	\$2,300.00	\$27,600.00
	4793	CX3020	12	\$2,575.00	\$30,900.00	\$1,930.00	\$23,160.00
	4973	MX3020	12	\$2,300.00	\$27,600.00	\$1,830.00	\$21,960.00
	Customer 104	Subtotal	36		\$96,600.00		\$72,720.00
Transaction Totals				349	\$760,225.00		\$641,670.00
Profit: 18.5%							

Figure 11-2 The Sample Report *Salesrpt*

## Report Design

To prepare a report, you want to consider:

- Which information from a table you want to use
- How you want major sections of your report to look
- Where you want your information to appear in each section
- Which subheadings and subfootings you want

With R:base REPORTS as your assistant, there are four basic steps you perform to create your report.

1. Design the format of your report. It is very useful to do this on paper so you can identify the table columns to be used and what variables you need to define. Use the *Edit* option to enter your report page and column headings, and any additional text material. (See the section "Editing the Report Format.")
2. Define any variables you need for the report. You know what variables you need from your paper report design. For example, if you want to show a total on the report which is not carried in the table you are reporting, you need to define a variable expression to compute that total. You can also use variables to retrieve information from other tables and to define date, time, or page numbers on your report. (See the section "Defining, Changing, and Deleting Report Variables.")
3. Indicate where the table columns and variables are to appear on the report. (See the section "Locating Columns and Variables on the Report.")
4. Identify the lines that correspond to the report heading, detail lines, subheadings, subfootings, and footing. If you have designed your report on paper and understand the purpose of each of the report sections, the *Mark* option enables you to easily indicate each report section. (See the section "Marking Heading, Detail, Breakpoint, and Footing Lines.")

## CREATING THE REPORT

To begin creating your report, type *REPORTS* at the R:base prompt as shown in the screen sample below:

```
R>REPORTS
Begin R:base REPORTS definition
Enter REPORT name:
```



## Choosing a Report Name

Next, you are asked to enter a report name. The name you choose can be between one and eight characters long. Choose a report name that you can easily associate with the table you use in the report. In the sample report used to illustrate this chapter, the report is called *salesrpt* for “sales report” and the corresponding table is called *transx* for “transactions.”

After entering a report name, you are asked to enter the name of the table to be used in the report.

An alternative method is to enter the report name immediately following the REPORTS command. If the report exists, the REPORTS menu is displayed. If the report does not already exist, R:base prompts for the table as shown on the following screen:

```
R> REPORTS salesrpt
Begin R:base REPORTS definition
Enter table name:
```

Only a single table is specified for each report. You can, however, request columns from other tables to appear on the report. (“Defining Columns from Other Tables” in this chapter.)

After you enter the name of the table, the REPORTS menu appears on your screen with a list of options to use for creating your report.

```
---Edit report---Define---Locate---Mark---Set---Help---Quit-----
```

The bottom of the screen contains help text, a list of the options with a brief statement explaining their purpose. You can obtain on-line help by either selecting the Help option or by pressing the [F10] function key. See the “Help for Report Generation” section.

Table 11-3 describes the options. To use the menus, press the space bar or an arrow key to highlight the selection you want. When your choice is highlighted, press [ENTER] to select it. Alternatively, you can select an option by pressing the key corresponding to the first letter (*E* for *Edit*, for example). All menus in REPORTS operate this way.

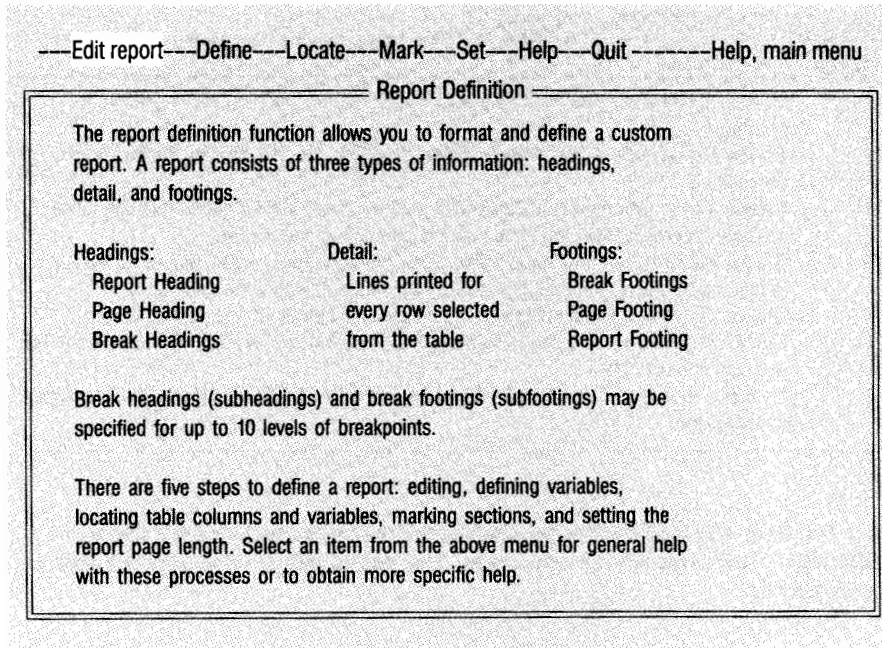
Table 11-3 REPORTS Menu Options

Option	Description
<b>E</b>	Enters <i>Edit</i> where you type in text for information such as report column headings
<b>D</b>	Enters <i>Define</i> where you define variables which are to be used during report processing
<b>L</b>	Enters <i>Locate</i> where you indicate the positions and field lengths for the table columns and variables which are to be printed on the report
<b>M</b>	Enters <i>Mark</i> where you define the types for each line of the report—heading, detail, or footing—and specify breakpoint subheadings and subfootings
<b>S</b>	Enters <i>Set Page Length</i> where you indicate the length of the report page
<b>H</b>	Enters <i>Help</i> where you can obtain on-line information for each report formatting step (or press [F10])
<b>Q</b>	Prompts to save or discard changes and exit from REPORTS or return to report processing

Once you have specified a table for your report, you may want to see a list of the table columns and variables to help design your report layout. Press the [F3] key to display this list.

## Help for Report Generation

On-line help is available for creating or updating your report. The help text for report definition offers both instructions and explanations for defining a report. To view this text, select *Help* from the REPORTS menu. You see the following information:



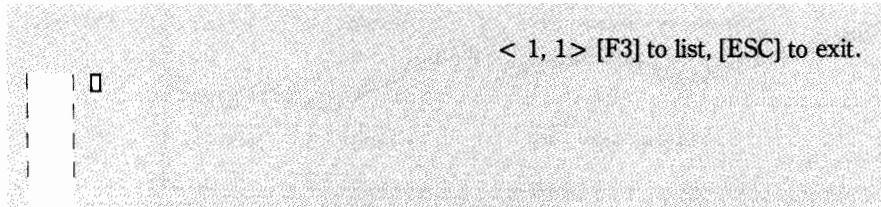
To display specific help information on one of the report options (*Edit*, *Define*, *Locate*, *Mark*, or *Set*), use the space bar or cursor keys to highlight your selection and then press [ENTER] to display the help screen for that option. When you are done, select *Quit* to return to the REPORTS menu.

## EDITING THE REPORT

### Editing the Report Format

After you have specified the report name and the table for the report, press [E] from the REPORTS menu to edit the new or existing report. You may return to *Edit* as often as you like when developing a report or changing an existing report format.

REPORTS displays the edit screen. If you have already worked on the report, some of the report information appears on the screen. If this is the first option you are using on a new report, the screen is blank except for the *Edit* screen heading. For a new report, the screen appears as follows:



The numbers displayed between the angle brackets (< >) show you the screen line and column of the cursor location. If you want to list the variables and table columns, press the [F3] function key. Press [ESC] to leave report editing. The highlighted area on the left is used to show the report sections (heading, footing, or detail).

### THE REPORT SECTIONS

A report is usually divided into three sections—heading, detail, and footing. Each of these sections are defined when you *Mark* the report. See “Marking Heading, Detail, Footing, and Breakpoint Lines” for details.

The headings contain information you enter such as the report title and column headings. The detail section is normally reserved for the information drawn from the table and defined variables which you *Locate* on your report. The footing contains information which is to appear at the end of the page or at the end of the report.

Figure 11-3 shows the major report sections on the sample report.

COMPATIBLE COMPUTER COMPANY DAILY SALES TRANSACTIONS								Date 03/11/85
								Page 1
Cust#	Customer Name	Receipt#	Product#	Units	Unit Price	Ext. Price	Cost	Ext. Cost
100	PC Distribution Inc.		129	Ernest Hernandez				
		4790	CX3000	15	\$1,800.00	\$27,000.00	\$1,530.00	\$22,950.00
		4796	PB3050	8	\$3,000.00	\$24,000.00	\$2,300.00	\$18,400.00
		5081	PX3050	5	\$3,000.00	\$15,000.00	\$2,350.00	\$11,750.00
		5080	MX3010	10	\$1,700.00	\$17,000.00	\$1,630.00	\$16,300.00
		5080	PB3040	15	\$2,250.00	\$33,750.00	\$2,175.00	\$32,825.00
Customer: 100				Subtotal	53	\$116,750.00		\$102,025.00
Transaction Totals								349
						\$760,225.00		\$641,670.00
Profit: 18.5%								

Figure 11-3 The Major Report Sections

### KEYBOARD FUNCTIONS

The *Edit* functions include the use of insert and delete characters, insert and delete lines, and cursor movement. Table 11-4 lists the keyboard functions available in the *Edit* option.

Table 11-4 Edit Keyboard Functions

Key	Definition
[Ins]	Inserts a space at the cursor position
[Del]	Deletes a space/character at the current cursor position
[→]	Moves the cursor right one space/character
[Ctrl] [→]	Moves the cursor to column 131
[←]	Moves the cursor left one space/character
[Ctrl] [←]	Moves the cursor to column 1
[↓]	Moves the cursor down one line
[↑]	Moves the cursor up one line
[PgUp]	Moves the cursor up 20 lines
[PgDn]	Moves the cursor down 20 lines
[Home]	Moves the cursor to the top of the report
[End]	Moves the cursor to the bottom of the report
[⇨]	Moves the cursor to the next tab setting*
[Shift] [⇨]	Moves the cursor to the previous tab setting*
[F1]	Inserts a line at the current cursor position
[F2]	Deletes the line at the current cursor position
[F3]	Displays the current variables and table columns
[F10]	Displays on-line help
[ESC]	Exit from editing

\*Tabs are automatically set at screen columns 10, 20, 30... up to column 130.



Although only 78 screen columns appear at one time, your report may be up to 131 columns wide. When you use the right arrow cursor key to continue past the 78th screen column, the screen shifts to let you see the 78 screen columns on the right side of the report.

You may place heading and text information on up to 66 lines. It is, therefore, possible to duplicate a typical 8-1/2 by 11 business form.

When you want to enter text, position the cursor and begin typing. See table 11-4 for the available editing functions.

## Defining, Changing, and Deleting Report Variables

Variables are useful if your report includes information which you cannot access directly from a table. You can use these variables to compute subtotals, totals, averages, percentages, or any other amounts needed in your report. REPORTS also provides the date and time, and the ability to number pages.

Global variables created with the SET VARIABLE command can be included in your report by defining a report variable equal to the value of the global variable. Precede the global variable with a period or dot ( . ), for example, *rptvar* = *.global*. System variables can be used by setting a report variable equal to the system variable. For example, *rptdate* = *#DATE*.

Another report variable is the lookup variable. This allows you to use columns from tables other than the specified report table in your report.

You may define up to 40 variables per report including those equivalent to global variables. The only restriction is that variable expressions must refer to table columns, another report variable, a constant value, a system variable, or a global variable.

By selecting the *Define* option from the REPORTS menu, you can either define, redefine, delete, reorder, or change the data type of a variable. This section describes each of these activities.

### DEFINING NEW VARIABLES

To define the variables used in the sales transactions report, select *Define* from the REPORTS menu. The following variable definition screen appears:

At the *Expression* prompt, enter the variable definition (or expression). Be sure to press [ENTER] at the end of the definition. If you make a mistake, you can correct it by using the back space key to erase the expression back to the error then retyping before you press [ENTER]. If you have already pressed [ENTER], you can reenter the expression correctly. See “Changing Existing Variables” in this chapter. Press [F10] to display on-line help for the *Define* option.

You may enter a variable in any of the forms shown in table 11-5. You must enter a space between each part of the expression. For example, *varname* = *value-colname* is not valid; *varname* = *value* - *colname* is valid. The operators are listed in table 11-6.

Table 11-5 Variable Formats

VARIABLE NAME	=	1ST OPERAND	OPERATOR	2ND OPERAND
varname	=	colname	op	colname
varname	=	colname	op	value
varname	=	value	op	value
varname	=	value	op	colname
varname	=	value		
varname	=	#DATE		
varname	=	#PAGE		
varname	=	#TIME		
varname	=	.global*		
varname	=	.global*	op	colname
varname	=	.global*	op	value
varname	=	.global*	op	varname2
varname	=	colname IN tblname WHERE conditions†		

\*A "dotted" variable is the value of a global variable defined with the SET VARIABLE command.

†This form of variable expression is used to retrieve columns from tables other than the table specified for this report.

Table 11-6 Variable Operators

Operator	Description
+	Adds the second operand value to the first operand value (or to combine two text values)
SUM OF	Computes the sum of all values of a table column and places that value in the variable
-	Subtracts the second operand from the first operand
/	Divides the first operand by the second operand
x	Multiplies the first operand by the second operand
%	Calculates the percentage of the first operand to the second operand (for example, 10 % 250 evaluates to 25)
&	Combines the operands to form a single text string with a space between the two values

As you define a report variable, REPORTS checks the expression for the correct form and alerts you to errors. If it is not sure of the variable data type, it requests confirmation when the variable is defined. You can select the correct data type from the displayed data type options. If you make a mistake in choosing the correct data type, you can use the TYPE function of *Define* to change a variable data type. See "Changing Variable Data Types" in this chapter.



SUM OF is the sum of all the specified table columns for all qualified rows (see "Printing the Report" in this chapter). For example, the total units sold, variable *totunit*, is defined as *totunit* = SUM OF *units*.

In REPORTS, null values in an expression are treated as if they were equal to zero. If an overflow condition occurs, the result is a NULL value. Overflow conditions are:

- Integers greater than 2,147,483,647
- Dollar amounts 100 trillion and over
- Real numbers outside the range  $\pm 9^{37}$

The following guidelines apply to report expressions:

- Expressions with DOLLAR data types always yield DOLLAR data types. These are rounded to the nearest penny. The following expressions are valid:

DOLLAR + DOLLAR	REAL X DOLLAR	INTEGER X DOLLAR
DOLLAR - DOLLAR	REAL % DOLLAR	INTEGER / DOLLAR
DOLLAR X DOLLAR	DOLLAR X REAL	DOLLAR X INTEGER
	DOLLAR / REAL	

- Expressions with DATE data types always yield INTEGER data types (days). The following expressions are valid:

DATE - INTEGER	DATE + INTEGER	INTEGER + DATE
----------------	----------------	----------------

### Calculating Report Subtotals

In your reports you may want to use variables to compute subtotals in the report. Subtotals are useful if you need totaled amounts, such as subtotals of purchases by customer, for each value in a specific table column. Each time a value changes in the table column, REPORTS prints a totaled amount for that value. The sample report has subtotals on the number of units (*SUM OF units*), the total extended price (*SUM OF extprice*), and the total extended cost (*SUM OF extcost*).

To define the variables which calculate these subtotals, you need the following definitions:

9:DOLLAR	:	subprice = SUM	OF extprice
10:INTEGER	:	subunits = SUM	OF units
11:DOLLAR	:	subcost = SUM	OF extcost

As you enter each expression and press the [ENTER] key, the variable you have defined is preceded by a number and the variable data type. The variable examples in this section are from the sample report. Press [F3] if you need a list of columns within the table used in the report.

When calculating subtotals, use the expression form  $x = x + y$  or  $x = \text{SUM OF } y$  to continuously add a value to the defined variable. Each time subtotal variables are printed, they must be reset to clear the current subtotal and prepare the variable for a new series of values. Resetting is specified when you set breakpoints. See “Marking Heading, Detail, Breakpoint, and Footing Lines” in this chapter.

The subtotals on the example report are all based on breakpoints. Subtotals or other calculations defined by variables may also be located across the report columns in the detail lines. You may have as many report columns as you can fit on the report.

You can also use a variable to count rows. For example, you can include a total number of transactions for each customer by defining a variable as follows:

```
tranctr = tranctr + 1
```

Then, you would reset the counter to zero at the customer breakpoints. See “Breakpoint Subheadings and Subfootings Option” in this chapter.

### Calculating Report Totals

At the end of a report, you may want to use variables to compute overall totals. Totals are useful if you need a totaled amount, such as costs or expenses, for an entire cost category or organization.

The totals calculated in the example report are total units, total extended price, and total cost. To calculate these totals, you need to add up the rows of each respective table column. The expressions below compute these amounts.

```
2:DOLLAR : extcost = cost      x units
3:DOLLAR : totcost  = SUM      OF extcost
4:DOLLAR : extprice = price     x units
5:DOLLAR : totprice = SUM      OF extprice
6:INTEGER : totunits = SUM      OF units
```

The variable *cost* is a value which must be found in a different table. See “Defining Columns from Other Tables” for an explanation of how this variable is defined. Do not reset overall total values at breakpoints.

### Calculating Percentages

You may also use variables to calculate percentages. There are two methods to calculate percentages. One is to use the expression format *varname* = *operand* % *operand* where *operand* is a table column, a defined variable, a set value, or a global variable. This method should be used when calculating percentages on detail lines.

The % operator can only function with REAL or INTEGER data types for the left side of the expression. You can, however, use a DOLLAR data type on the right side of the expression. The answer is returned as a dollar amount, not as a percentage.

REPORTS prompts for the correct data type as well. You have the option to print the percentage as a REAL number or as an INTEGER. For example, to calculate the percentage of cost versus price, you need to use the second method described below.

The second method involves a series of calculations using variables which hold calculated values but do not appear on the report. For example, in the Daily Sales Transaction Report, the profit percentage is reported at the end of the report. The profit percentage is derived by the following series of variable expressions.

```
7:DOLLAR      : profit1      = totprice — totcost
8:REAL        : profit2      = profit1 / totcost
9:REAL        : profit       = profit2 x 100
```

When the profit is calculated, REPORTS first calculates the expressions for *profit1* and *profit2* which are not printed on the report. Then, *profit* is calculated as the final step and printed at the bottom of the report.

### Defining Columns from Other Tables

To use columns in your report which exist in tables other than the table assigned to the report, you need to define a variable to hold the other table's column value. This variable is referred to as a *lookup variable*. Use the following definition format at the *Expression* prompt:

```
varname = column IN table WHERE condlist
```

In the sample report, the sales representative's first and last names are held in the *salesrep* table but the report is based on the *transx* table. To get the employee's first and last names, the variables *fname* and *laname* are set as lookup variables:

```
1:DOLLAR      : cost          = cost IN product WHERE prodid = prodid
15:TEXT       : fname         = firstname IN salesrep WHERE empid = empid
16:TEXT       : laname        = lastname IN salesrep WHERE empid = empid
17:TEXT       : slsname       = fname & laname
18:TEXT       : custname      = company IN custlist WHERE custid = custid
```

The first variable, *cost*, is a column from the *product* table used to obtain the cost of each product in the report. It is placed at the beginning since the product cost is required to calculate other variables used in the report such as extended cost.

REPORTS knows that *prodid* in the phrase *WHERE prodid* refers to the column in the *product* table and that *prodid* in the phrase *= prodid* refers to the table assigned to the report.

The next two variables are used to derive the sales representative's name from the *salesrep* table using the employee number as the basis of the lookup. The variable *fname* is defined as the column *firstname* in table *salesrep* where the table column *empid* is equal to the current employee number. The last name is derived in the same way.

In the sample report, the employee name *slname* is printed in the report subheading.

Variables defined as columns from other tables are not limited to 30 characters like text constant variables. See "Advanced Report Concepts" for information on using variables defined from columns of other tables to produce form letters.

R:base can usually perform much faster if the column from the other table is a KEY column. If you want faster performance when accessing another table, you may wish to build a key for the column you need (see the BUILD KEY command in chapter 6).

Note that when you use a WHERE clause in the definition, the first part (*WHERE x*) must refer to a column in the other table. The second part following the operator can be a column in the report table, a variable, a global variable, or a constant. All of the following are legitimate:

```
WHERE colname1 = colname2
WHERE colname GT variable
WHERE colname CONTAINS .global
WHERE colname LE 100
WHERE colname FAILS
WHERE colname EXISTS
```

Up to ten conditions can be stated in the WHERE clause. For example:

```
WHERE colname1 EQ Smith AND colname2 EQ Boston OR colname3 LE $10,000
```

This WHERE clause uses three conditions. The first two are separated by AND so both conditions must be met (the value of *colname1* must be Smith and the value of *colname2* must be Boston) or the third condition must be true (the value of *colname3* must be less than or equal to \$10,000). See the *R:base 5000 Reference Manual* for full information on the WHERE clause.

You do not need a direct reference to another table in the report table to use the lookup feature. For example, if the report table contained a column which matched a second table's column, and the second table contained a column which matched yet a third table, you can define two lookup definitions. The first would retrieve the second table's reference to the third table and the second would retrieve the actual data from the third table. Suppose the three tables looked like this:

<u>transdet</u>	<u>order</u>	<u>customer</u>
<i>detailid</i>	<i>orderid</i>	<i>custid</i>
<i>orderid</i>	<i>custid</i>	<i>custname</i>

If *transdet* is a detail transaction file used as the report table, *order* the main order table, and *customer* is the customer master table then you define the following to retrieve the customer name:

```
custno = custid IN order WHERE orderid = orderid
custname = custname IN customer WHERE custid = custno
```

*Custno* is used temporarily to retrieve the customer number from the second table *order* and then the customer name *custname* is retrieved from the third table *customer* using *custno*.

### Defining Date, Time, and Page Numbers

To use the date, time, or page numbering options, you need to define variables to hold the values. Be sure not to use any of the R:base reserved words listed in chapter 5. Use the following general formats, substituting your own variable names: *varname* = #xxx where xxx is DATE, TIME, or PAGE. The sample report uses the date and page number as shown below.

```
13:DATE      : rptdate  = #DATE
14:INTEGER   : rptpage  = #PAGE
```

Once you have defined a variable to hold the date, you can calculate other information using the system date and dates in the table columns. For example, you can determine a due date by adding days to a date. To do this, you can define a variable similar to the following:

```
duedate = tdate + 30
```

Similarly, you can subtract an integer number of days from either the system date or a date in a table column. You cannot, however, subtract one date from another to find the number of days between them. Date arithmetic in REPORTS is limited to adding or subtracting an integer from a defined DATE report variable.

## Combining Text Expressions

You can define a variable to store a combined text variable from text table columns, text variables, or a specific text value you enter. For example, suppose you have first names and last names held in separate table columns. In the sample table *salesrep*, the employee's first and last names are held as separate columns. To print the first and last names together, you define a variable to contain both. To do this, at the *Expression* prompt type:

```
slsname = fname & lname
```

The ampersand (&) combines the values in the two table columns and places a single space between them. To combine text values without the additional space, use a plus sign (+) instead of the ampersand. The maximum length of a combined text variable is 1500 characters. If a constant text value is used, the maximum length of the text constant is 30 characters.

You can combine any text value with any other text value. You can, for example, combine a table column with a variable, two variables, a table column with a text value in quotes, a variable with a text value in quotes, or two text values in quotes. All of the following forms are allowed as long as all of them are of the TEXT data type:

```
varname = "text string 1" & "text string 2"
varname = column + "text string"
varname = variable & column
varname = variable + "text string"
```

## Using Constants in Expressions

A constant is a set value. It can be any data type—TEXT, INTEGER, REAL, DOLLAR, DATE, or TIME. TEXT type constants are limited to 30 characters. When you use a numeric constant in an expression, you simply enter it as a number:

```
varname = cost x 100
varname = 101.22
varname = $100.00
```

If using a TEXT data type, use quotation marks. For example:

```
varname = "Constant Text"
```

If using a DATE or TIME data type, enter the date or time in any legal R:base format. For example:

```
varname = 09:15:00
varname = 03/01/85
```

### Using Global Variables in Expressions

Global variables are variables created by the SET VARIABLE command. In REPORTS, global variables can be used in a report variable expression.

A global variable must be preceded by a period to distinguish it from a report variable. Following are examples of ways you can use a global variable:

```
varname = .global  
varname = .global + 100  
varname = colname x .global
```

### Setting Printer Control Variables

Some printers are capable of compressed or expanded print but require a special control sequence before changing print modes. You can define variables to act as printer control sequences. Use the following format to define printer control sequences in a variable:

```
prtvar = <print control list>
```

*Prtvar* is the name of the variable and *print control list* is the required printer control sequence for the function you want. This is entered as ASCII values. ASCII, which means *American Standard Code for Information Interchange*, is a coding method where standard keyboard characters such as letters, numbers, punctuation, or special codes are assigned a numeric value which is understood by most computers. Once you have defined your printer control sequences as variables, you then use the *Locate* option to place these variables in the body of the report.

You can place printer control variables in your report format just like any other variable. They do not print any control characters on the report, they only change the print characteristics of the text following their location. For example, to print your entire report in compressed print, define a variable containing the compress print control codes and then locate that variable on the top line of the report. Also, define a variable to change the print back to normal size and then locate that variable on the last line of the report following all other variables and columns. See "Locating Table Columns and Variables on the Report" for information on how to place variables on a report.

For example, the following printer control sequence sends the ASCII equivalent of the escape key (27) followed by the ASCII equivalent for the minus sign (45). An Epson printer interprets this code as set or cancel underlining.

```
prtunder = <27 45>
```

See your printer user's manual for the printer control sequences and an ASCII character table.

## CHANGING EXISTING VARIABLES

You can easily change an existing variable expression by entering the new expression using the existing variable. REPORTS recognizes that the variable name already exists and prompts:

```
Variable exists. Redefine it (Y/N)?
```

Press [Y] to redefine the variable with the new expression. Press [N] if you do not want to change the existing variable expression.

## REORDERING, DELETING, AND DATA TYPING VARIABLES

Once you have specified your report variables, you may want to reorder, delete, or change the data type of a variable.

### Reordering Variables

The variable order is important, particularly when subheadings and subfootings are used and variables are reset after printing. An incorrect total on your report probably means that one or more of your variables are out of order. For example, suppose you have the following variables defined in this order:

```
1:DOLLAR : totcost = totcost + extcost
2:DOLLAR : extcost = cost x units
```

You may have an incorrect total value, because *extcost* is required to calculate *totcost* but is not defined until after it is used. To change the order, use the REORDER command at the *Expression* prompt as follows:

```
REORDER varname position
```

This tells REPORTS to move the variable *varname* to the position indicated. To reorder *extcost* to the first position, type:

```
REORDER extcost 1
```

### Deleting Variables

If you want to delete a report variable, use the DELETE command at the *Expression* prompt as follows:

```
DELETE varname
```

### Changing Variable Data Types

If you want to change the specified data type of an existing variable, at the *Expression* prompt enter TYPE, the name of the variable, and the new data type:

```
TYPE varname vartype
```



**EXAMPLE REPORT VARIABLES**

The following screen shows the entire list of variables defined for the example report:

Expression:		[F3] to list, [ESC] to exit	
1:DOLLAR	: cost	= cost	IN product WHERE prodid = prodid
2:DOLLAR	: extcost	= cost	x units
3:DOLLAR	: totcost	= SUM	OF extcost
4:DOLLAR	: extprice	= price	x units
5:DOLLAR	: totprice	= SUM	OF extprice
6:INTEGER	: totunits	= SUM	OF units
7:DOLLAR	: profit1	= totprice	- totcost
8:REAL	: profit2	= profit1	/ totcost
9:REAL	: profit	= profit2	x 100
10:DOLLAR	: subprice	= SUM	OF extprice
11:INTEGER	: subunits	= SUM	OF units
12:DOLLAR	: subcost	= SUM	OF extcost
13:DATE	: rptdate	= #DATE	
14:INTEGER	: rptpage	= #PAGE	
15:TEXT	: frname	= frstname	IN salesrep WHERE empid = empid
16:TEXT	: laname	= lastname	IN salesrep WHERE empid = empid
17:TEXT	: slsname	= frname	& laname
18:TEXT	: custname	= company	IN custlist WHERE custid = custid

**Locating Table Columns and Variables on the Report**

*Locate* helps you to create the layout of your report. You can specify where you want the table column and variable values to appear. When you select the *Locate* option from the REPORTS menu, REPORTS gives you the choice of locating new table columns and variables or relocating, changing, or deleting locations of table columns and variables which have already been located.

The *Locate* menu looks like this:

```

-----Locate-----Relocate-----Help-----Quit-----

```

- Select *Locate* to place new table columns and variables
- Select *Relocate* to change the location or delete currently located table columns and variables
- Select *Help* to display on-line help (or press [F10])
- Select *Quit* to redisplay the REPORTS menu (or press [ESC])

Press [F3] to display the table columns and variables available for location.

### LOCATING NEW VARIABLES AND TABLE COLUMNS

When you select *Locate* from the *Locate* menu, the *Locate* screen is displayed. If you have already used *Edit* to enter headings and other text data, the information is displayed. It is a good idea to enter report column headings before locating table columns and variables. If necessary, you can always return to *Edit* to adjust the report column heading positions to the locations of the table columns and variables.

The *Locate* screen below shows the report column headings for the sample transaction report.

Name of column or variable: <input type="text"/>		[F3] to list, [ESC] to exit				
		COMPATIBLE COMPUTER COMPANY				
		DAILY SALES TRANSACTIONS				
		Cust#	Customer Name	Sls#	Salesman Name	
			Receipt#	Product#	Units	Price
						Ext. Price
		-----				

REPORTS prompts for the variable or table column you want to locate. Enter the variable or table column name or press [F3] to display a list of the available table columns and defined variables. Press [ESC] to exit from variable or table column location.

After you enter the variable or table column name, the cursor is shown on the screen. It is either in the upper-left corner of the report or wherever its last position was at the end of your last task. Use the arrow keys to place the cursor where you want to locate the variable or table column.



The control keys you can use are shown in table 11-7.

Table 11-7 Locate Control Keys

Key	Description
[→]	Moves the cursor right one space/character.
[←]	Moves the cursor left one space/character.
[↓]	Moves the cursor down one line.
[↑]	Moves the cursor up one line.
[⇨]	Moves the cursor to the next tab setting.*
[Shift] [⇨]	Moves the cursor to the previous tab setting.*
[Home]	Moves the cursor to the top of the report.
[End]	Moves the cursor to the bottom of the report.
[PgUp]	Moves the cursor up 20 lines.
[PgDn]	Moves the cursor down 20 lines.
[F3]	Displays the current variables and table columns.

\*Tab settings are at screen columns 10, 20, 30 . . . up to column 130.

Press [S] to mark the starting point for the table column or variable you are locating. REPORTS moves the cursor to the right-most edge of the report column based on the data type of the variable or table column. If the table column is a TEXT data type, REPORTS places the cursor one space to the right of the S regardless of the text length.

Use the arrow keys again to move the cursor to where you want the last character position of the variable or table column value to appear. You can make the width smaller than REPORTS suggests but remember to leave enough room on dollar amounts for the dollar sign and commas.

Press [E] to mark the right side of the report column location.

REPORTS prompts for another variable or table column. Continue to locate the variables and table columns you want in your report. When you are done, press [ESC] to return to the REPORTS menu.

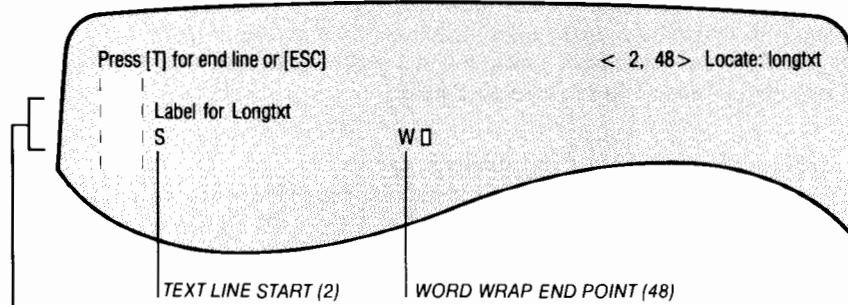
### Setting Text Wrap Locations

To mark locations for long text fields, you may want to set up a vertical report column for the text instead of a long horizontal field. Text wrap marking allows you to mark a vertical report column for long text and also to cut off the text (truncate) and to indent the first line of text.

If you have long text fields which you want to report as paragraphs, press [S] at the beginning point and press [W] to mark the ending point of the location. When you mark the end of the location with W, REPORTS prompts for truncation as shown in figure 11-4.

If you want the entire text value to appear on the report, press [ESC]. For example, if the text in this paragraph is the value for the table column *longtxt*, then the report appears as shown in figure 11-4 if you neither truncate nor indent the text.

#### WHAT YOU SEE WHILE IN REPORTS MODE:



#### REPORT RESULTS:

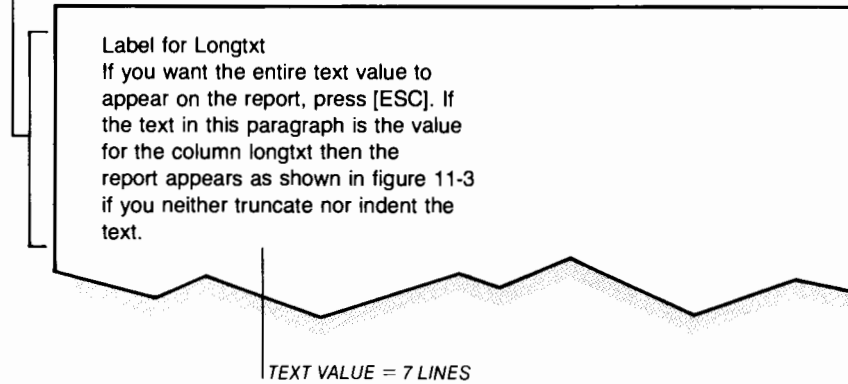
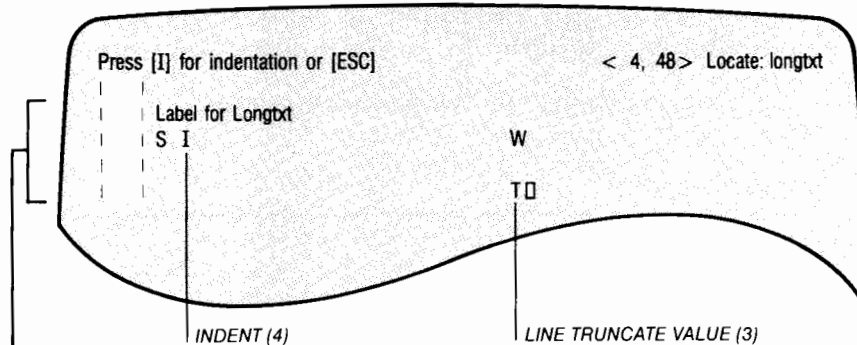


Figure 11-4 Non-Truncated Paragraph Text

To limit or truncate the text printed to a specific area, you need to mark the ending point of the text. To do this, move the cursor to the point which represents the last line and report column where you want the text to appear and press [T]. Any additional text is truncated. REPORTS breaks lines at the last available space. For example, if the remaining space is five spaces and the word is six characters long, the word is wrapped down to the next line or, if the text was truncated, does not appear on the report. Next, REPORTS displays the indentation prompt as shown in figure 11-5.

This prompt allows you to indent the first text line. Move the cursor to the indentation point and press [I]. If you do not want indentation, press [ESC]. You can mark an indentation to the right or left of the S marker. Figure 11-5 illustrates how this paragraph looks on a report if it is truncated at the third line and indented as shown in the screen example.

#### WHAT YOU SEE WHILE IN REPORTS MODE:



#### REPORT RESULTS:

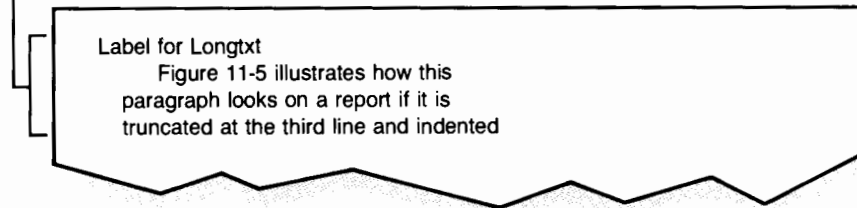


Figure 11-5 Truncated Paragraph Text

### Masks for DOLLAR Data Types

For special purposes, such as printing dollar amounts on checks, you may want to print dollar amounts in the following format:

\*\*\*\*\*\$10.00

Mark the location of the dollar table column or variable as you would mark any table column. If the table column or variable is a dollar amount, REPORTS prompts:

Print with check format (Y/N)?

Press [Y] to print the dollar amount in the format shown above. If you do not want to mask the dollar amount to this format, press [N] to print the dollar amount right-justified in the marked report column without leading asterisks.

### CHANGING AND DELETING LOCATIONS OF VARIABLES AND TABLE COLUMNS

If you need to change or delete a table column or variable location, select *Relocate* from the *Locate* menu. The *Locate* screen is displayed as shown below:

IF YOU SELECT

Keep	Change	Delete	Quit	Location for:custid
COMPATIBLE COMPUTER COMPANY				
DAILY SALES TRANSACTIONS				
Cust#	Customer Name	Receipt#	Product#	Units
Sls#	Salesman Name	Unit Price	Ext. Price	
S E S		S E S	E S E S	E S
		S E S	E S	E S

THIS INFORMATION DOES NOT CHANGE

REPORTS prompts you for each located table column and variable. You have these choices:

- *Keep* the variable or table column in the same location
- *Change* the location of the variable or table column
- *Delete* the variable or table column location
- *Quit* to the *Locate* menu

If you select *Keep*, REPORTS keeps the table column or variable in the current location and prompts for the next table column or variable. The order of the prompting depends on the order in which the variables and table columns were originally located.

If you select *Change*, REPORTS places the cursor at the specified variable or table column location. The S and E location markers disappear. Use the cursor keys to relocate the variable or table column to a new position. Use the [S] and [E] keys in the same manner as when locating a new variable (see “Locating New Variables and Table Columns”). When you have relocated the variable or table column, REPORTS prompts for the next variable or table column.

If you select *Delete*, REPORTS removes the variable or table column location markers and then prompts for the next variable or table column. You must use *Locate* rather than *Relocate* to reenter the location if you want the variable or table column to appear on your report.

If you select *Quit*, REPORTS redisplay the *Locate* menu. Select *Quit* from the *Locate* menu to exit to the REPORTS menu.

## **Marking Heading, Detail, Breakpoint, and Footing Lines**

With the *Mark* REPORTS option, you specify where you want your heading, detail, breakpoint, and footing lines to appear. You tell REPORTS which lines you want at the top and bottom of the report and which lines are for information. You can request up to ten breakpoints to create subheadings and subfootings.

A heading contains information such as the report title, column headings, date, time, and page number. There are two heading levels: report and page. A report level heading appears only once, at the beginning of the report. Since you can set a page eject following the report heading, it can be used as a title page for the entire report. The page heading appears at the top of every report page.

The detail section contains the information drawn from the table. It usually consists of the data that will be printed repeatedly on a report. However, you can include additional text information if you produce a form such as a check.

The footing contains information which is to appear at the bottom of the report, for example grand totals. It may also contain text information you enter through the report editor. Like the heading, the footing has two levels: report and page. The report footing appears only once at the end of the entire report. The page footing appears at the bottom of every page of the report.

Breakpoints are used to print subheading and subfooting information whenever a specified table column or variable changes value. If a subheading is indicated, then a form feed is optionally sent to the printer and the specified subheading information is printed on the page. When a subfooting breakpoint is indicated by a change in the value of the table column, the marked subfooting lines are printed. This allows you to specify up to ten subheadings and subfootings within a report.

To mark report lines, select *Mark* from the REPORTS menu. REPORTS displays this screen:

---Report---Page---Detail---Break---List---Help---Quit---

Table 11-8 Mark Menu Options

Option	Purpose
<i>Report</i>	Mark or re-mark report level heading and footing. Also set page ejects.
<i>Page</i>	Add or delete variables to be reset at the end of each page then mark or re-mark page level heading and footing lines.
<i>Detail</i>	Mark or re-mark detail lines.
<i>Break</i>	<ol style="list-style-type: none"> <li>1. Add or delete break table columns for each level breakpoint.</li> <li>2. Select page eject before break heading.</li> <li>3. Add or delete variables to be reset after break (subfootings must be defined variables). Each entered break table column represents a numbered breakpoint. The same table column can break for both a subheading and a subfooting (Hn and Fn).</li> <li>4. After entering the break table column, mark break subheadings, if any, by pressing [H] at the break heading lines. Then, mark break subfootings by pressing [F] at the break footing lines. Subtotals must be defined variables and should be reset.</li> </ol>
<i>List</i>	Display the current status of any headings, footings, detail lines, and break lines which are already marked.
<i>Help</i>	Display on-line help on the <i>Mark</i> option (or press [F10]. Once you enter <i>Help</i> , you can select the <i>Mark</i> option on which you need information.
<i>Quit</i>	End <i>Mark</i> and return to the REPORTS menu (or press [ESC]).



**REPORT HEADING AND FOOTING OPTION**

The *Report* option sets the heading and footing lines for the entire report. The report heading lines appear at the beginning of the report and the report footing lines appear at the end of the report. The *Report* option is also used to set page ejects (a form feed) for the report. You have options to eject a page before the report, after the report heading, before the report footing, and after the report footing.

1. Select *Report* from the *Mark* menu.
2. REPORTS prompts:

Do you want to remove the initial carriage return[NO]?

Press [Y] to inhibit the initial carriage return when the report is printed. Press [ENTER] to allow the initial carriage return. If you remove the initial carriage return, the printer will start printing the report at the current print head position which may change the report heading format.

3. REPORTS prompts with the following four queries:

Page eject before report[NO]?  
Page eject after report heading[NO]?  
Page eject before report footing[NO]?  
Page eject after report footing[NO]?

The bracketed words show the default or current status. Press [Y] if you want a page eject at the specified points during report printing. Press [N] or just press [ENTER] if you want no page eject.

4. If this is the first time you have requested report marking, REPORTS displays the report screen and requests that you press [H] to mark the report heading lines.

If report headings were marked previously, REPORTS prompts:

Re-mark lines for report heading[YES]?

Press [ENTER] to accept the default answer *YES*. Press [N] to leave the report heading lines the way they are.

5. If report heading lines were not previously marked or you answered *YES* to the *Re-mark* prompt, the cursor is moved into the highlighted column at the left margin of the report. Use the cursor keys to move to the lines you want to mark. Press [H] at the lines you want as heading lines. REPORTS displays *HR* in the marked columns. Press [ESC] when you have marked all report heading lines.
6. If report footing lines were previously marked, REPORTS prompts:

**Re-mark lines for report footing [YES]?**

Press [ENTER] to accept the default answer *YES*. Press [N] to leave the report footing lines the way they are.

7. If you answered *YES* to the *Re-mark* prompt or report footing lines were not previously marked, REPORTS prompts:

**Press [F] to mark report footing:**

Use the cursor keys to move to the lines you want to mark. Press [F] at the lines you want as footing lines and press [ESC] when you have marked all footing lines.

8. REPORTS redisplay the *Mark* menu.

You will see a screen like the one shown in figure 11-6 when the report headings are marked. Figure 11-6 also illustrates the report heading and footing lines as printed on the sample report.



## WHAT YOU SEE WHILE IN REPORTS MODE:

Press [H] to mark report heading: 4 [ESC] to exit

COMPATIBLE COMPUTER COMPANY  
DAILY SALES TRANSACTIONS

Cust#	Customer Name	Receipt#	Product#	Units	Unit Price	Ext. Price
S	E	S	E	S	E	S
S	E	S	E	S	E	S
Customer S E Subtotals				S	E	S
Transaction Totals				S	E	S
Profit: S				E%		

## REPORT RESULTS:

COMPATIBLE COMPUTER COMPANY DAILY SALES TRANSACTIONS							Date 03/11/85
							Page 1
Cust#	Customer Name	Receipt#	Product#	Units	Unit Price	Ext. Price	Cost
100	PC Distribution Inc.	129	Ernest Hernandez				
	4790	CX3000	15	\$1,800.00	\$27,000.00	\$1,530.00	\$22,950.00
	4796	PB3050	8	\$3,000.00	\$24,000.00	\$2,300.00	\$18,400.00
	5081	PX3050	5	\$3,000.00	\$15,000.00	\$2,350.00	\$11,750.00
	5060	MX3010	10	\$1,700.00	\$17,000.00	\$1,630.00	\$16,300.00
	5080	PB3040	15	\$2,250.00	\$33,750.00	\$2,175.00	\$32,625.00
Customer 100 Subtotal				53		\$116,750.00	\$102,025.00
Transaction Totals				349		\$760,225.00	\$641,670.00
Profit: 18.5%							

Figure 11-6 Report Heading and Footing Lines

## PAGE HEADING AND FOOTING OPTION

The *Page* option sets the page heading and footing lines. The page heading is printed at the top of each report page. The page footing is printed at the bottom of each report page. You also indicate which variables are to be reset at the end of each page printed.

1. Select *Page* from the *Mark* menu.
2. REPORTS displays:

```

Add Delete Quit ----- Variables reset at end of page
  
```

3. Select *Add* to add variables to be reset at the end of each page. Select *Delete* to remove variables from this list. Select *Quit* to continue to the marking phase.
4. In either case, REPORTS prompts for the variable name. Enter the name of a variable which you want reset at the end of each page. Enter the name of a variable already listed to delete it from the list. Press [F3] to display a list of available table columns and variables.
5. Select *Quit* when you are done adding or deleting variables to be reset. REPORTS then prompts for page heading and footing lines (see steps 3 to 5 in "Report Heading and Footing Option"). Either mark the heading and footing lines as described or press [ESC] to end the option.
6. If this is the first time you have requested page marking, REPORTS displays the report screen and requests that you press [H] to mark the page heading lines.

If page headings were marked previously, REPORTS prompts:

```

Re-mark lines for page heading[YES]?
  
```

Press [ENTER] to accept the default answer *YES*. Press [N] to leave the page heading lines as they are.

7. If page heading lines were not previously marked or you answered *YES* to the *Re-mark* prompt, the cursor is moved into the highlighted column at the left margin of the report. Use the cursor keys to move to the lines you want to mark. Press [H] at the lines you want as page heading lines. REPORTS displays *HP* in the marked columns. Press [ESC] when you have marked all page heading lines.
8. If page footing lines were previously marked, REPORTS prompts:

Re-mark lines for page footing[YES]?

Press [ENTER] to accept the default answer *YES*. Press [N] to leave the page footing lines as they are.

9. If you answered *YES* to the *Re-mark* prompt or page footing lines were not previously marked, REPORTS prompts:

Press [F] to mark page footing:

Use the cursor keys to move to the lines you want to mark. Press [F] at the lines you want as footing lines and press [ESC] when you have marked all footing lines.

10. REPORTS redisplay the *Mark* menu.

The sample report does not use page footing lines. Page heading lines are used for the column heads for the report and for the report page number. Figure 11-7 illustrates the results of the page heading lines marked in the sample report.

WHAT YOU SEE WHILE IN REPORTS MODE:

Press [F] to mark page footing: 9 [ESC] to exit

COMPATIBLE COMPUTER COMPANY  
DAILY SALES TRANSACTIONS

Cust#	Customer Name	Receipt#	Product#	Units	Unit Price	Ext. Price
100	PC Distribution Inc.	4790	CX3000	15	\$1,800.00	\$27,000.00
		4796	PB3050	8	\$3,000.00	\$24,000.00
		5081	PX3050	5	\$3,000.00	\$15,000.00
		5060	MX3010	10	\$1,700.00	\$17,000.00
		5080	PB3040	15	\$2,250.00	\$33,750.00
Customer Subtotal				53		\$116,750.00
Transaction Totals				349		\$760,225.00
Profit: 18.5%						

REPORT RESULTS:

COMPATIBLE COMPUTER COMPANY DAILY SALES TRANSACTIONS							Date 03/11/85	
							Page 1	
Cust#	Customer Name	Receipt#	Product#	Units	Unit Price	Ext. Price	Cost	Ext. Cost
100	PC Distribution Inc.		129	Ernest Hernandez				
		4790	CX3000	15	\$1,800.00	\$27,000.00	\$1,530.00	\$22,950.00
		4796	PB3050	8	\$3,000.00	\$24,000.00	\$2,300.00	\$18,400.00
		5081	PX3050	5	\$3,000.00	\$15,000.00	\$2,350.00	\$11,750.00
		5060	MX3010	10	\$1,700.00	\$17,000.00	\$1,630.00	\$16,300.00
		5080	PB3040	15	\$2,250.00	\$33,750.00	\$2,175.00	\$32,625.00
	Customer 100	Subtotal	53			\$116,750.00		\$102,025.00
Transaction Totals				349		\$760,225.00		\$641,670.00
Profit: 18.5%								

Figure 11-7 Page Heading Lines in the Sample Report

### DETAIL OPTION

The *Detail* option is used to indicate the detail lines on the report.

1. Select *Detail* from the *Mark* menu.
2. If detail lines were previously marked, REPORTS prompts:

Re-mark lines for detail [YES]?

Press [ENTER] to accept the default answer *YES*. Press [N] to leave detail lines as they are.

3. If you answered *YES* to the *Re-mark* prompt or detail lines were not previously marked, REPORTS displays the report and prompts:

Press [D] to mark detail lines:

Move the cursor to the lines you want to mark as detail lines. Press [D] at these lines. REPORTS displays a *D* in the left column. Press [ESC] when you are done marking detail lines.

4. REPORTS redisplay the *Mark* menu.

The sample report is marked as shown in figure 11-8.

WHAT YOU SEE WHILE IN REPORTS MODE:

Press [D] to mark report detail: 12 [ESC] to exit

COMPATIBLE COMPUTER COMPANY  
DAILY SALES TRANSACTIONS

HP	Cust#	Customer Name	Sis#	Salesman Name	Receipt#	Product#	Units	Unit Price	Ext. Price
HP									
HP									
HP									
D	S	E	S			E	S	E	S
D					S	E	S	E	S
FR									
FR									
FR	Profit: S	E%							

REPORT RESULTS:

COMPATIBLE COMPUTER COMPANY										
DAILY SALES TRANSACTIONS										
									Date 03/11/85	
									Page 1	
Cust#	Customer Name	Sis#	Salesman Name	Receipt#	Product#	Units	Unit Price	Ext. Price	Cost	Ext. Cost
100	PC Distribution Inc.	129	Ernest Hernandez							
	4790	CX3000	15				\$1,800.00	\$27,000.00	\$1,530.00	\$22,950.00
	4796	PB3050	8				\$3,000.00	\$24,000.00	\$2,300.00	\$18,400.00
	5081	PX3050	5				\$3,000.00	\$15,000.00	\$2,350.00	\$11,750.00
	5060	MX3010	10				\$1,700.00	\$17,000.00	\$1,630.00	\$16,300.00
	5080	PB3040	15				\$2 250.00	\$33,750.00	\$2,175.00	\$32,625.00
	Customer 100	Subtotal	53					\$116,750.00		\$102,025.00
				Transaction Totals		349		\$760,225.00		\$641,670.00
Profit: 18.5%										

Figure 11-8 Detail Lines on the Sample Report



**BREAKPOINT SUBHEADINGS AND SUBFOOTINGS OPTION**

The *Break* option sets the subheadings and subfootings for the report. You can request a page eject before subheadings, if desired. If a page eject is requested, the page headings are printed on the new page above the subheading. You also indicate variables which are to be reset after a subheading or subfooting is printed.

1. Select *Break* from the *Mark* menu.
2. REPORTS prompts:

Do you want to manually reset break variables[NO]?

Press [Y] to allow REPORTS to reset lower level break variables. Press [ENTER] to reset lower level variables yourself. You still need to reset the variables for the specific break level. See step 6.

3. If you have previously entered breakpoints, REPORTS prompts:

Break number, 1-n[ n]:

where *n* is the next available breakpoint level. If you want to change a previously entered breakpoint, press the number key corresponding to the breakpoint level you want to change. If you want to enter another breakpoint, press [ENTER] to default to a new breakpoint entry. If you do not want to change or enter breakpoints, press [ESC].

4. If you pressed [ENTER] in step 3 to add a breakpoint or if you have not previously entered breakpoints, REPORTS prompts:

Break column or variable:

Enter the table column or variable name you want to define as a breakpoint indicator. When the value of the table column or variable changes, the marked subheading and subfooting lines are printed. Continue to step 5.

If you entered an existing breakpoint number in step 3, REPORTS prompts:

Break column or variable, or DELETE(breakname):

where *breakname* is the name of the table column or variable set for this breakpoint. If you want to change the breakpoint column or variable name, type in the new name and press [ENTER]. Continue to step 5. If you want to delete this breakpoint, type DELETE and press [ENTER]. Continue to step 9.

5. REPORTS then prompts:

Page eject before break heading[NO]?

Press [ENTER] for no page eject before the break heading. Press [Y] to force a page eject before the indicated break heading.

6. REPORTS prompts with the option line shown in the following screen.

—Add—Delete—Quit—Variables to reset after break

- Select *Add* to add a variable which is to be reset after the breakpoint. For the sample report, the three subtotal variables must be reset—*subunits*, *subcost*, and *subprice*—plus any lower level variables if you answered *NO* in step 2.
- Select *Delete* to remove a variable from the list.
- Select *Quit* to continue to breakpoint marking.

7. REPORTS prompts:

Press [H] to mark break heading:

Place the cursor at the first line you want as the breakpoint subheading and press [H]. The letter *H* followed by the number of this breakpoint is displayed in the left column. Move the cursor to the next break line and press [H] and so on. Mark all break subheading lines for this breakpoint. When you are done, press [ESC] to continue.

## 8. REPORTS prompts:

Press [F] to mark break footing:

Place the cursor at the lines you want as the breakpoint subfooting lines and press [F]. The letter *F* and the number of this breakpoint are displayed in the left column. Mark all break subfootings for this breakpoint. When you are done, press [ESC] to end breakpoint marking.

9. REPORTS redisplay the *Break column or variable* prompt. Enter another table column or variable for a breakpoint or press [ESC] to end breakpoint setting.

10. REPORTS redisplay the *Mark* menu.

The screen example in figure 11-9 shows the break lines marked for the sample report.

The line marked *H1* and the three lines marked as *F1* are breakpoint lines. These lines are printed whenever the table column specified as the break table column changes. If, for example, the customer number is specified as the breakpoint table column, then each time the customer number changes, the subtotals for the last customer are printed.

Figure 11-9 also illustrates the breakpoint subheadings and subfootings on the sample report.

WHAT YOU SEE WHILE IN REPORTS MODE:

Press [F] to mark break footing: 15 [ESC] to exit

COMPATIBLE COMPUTER COMPANY  
DAILY SALES TRANSACTIONS

HP	Cust#	Customer Name	Sls#	Salesman Name			
HP		Receipt#	Product#	Units	Unit Price	Ext. Price	
HP							
H1	S E S		E S E S			E	
D		S E S	E S	E S		E S	E
F1		Customer S E Subtotal	S E			S	E
F1							
FR		Transaction Totals	==			==	
FR	Profit: S	E%					

REPORT RESULTS:

COMPATIBLE COMPUTER COMPANY DAILY SALES TRANSACTIONS							Date 03/11/85
							Page 1
Cust#	Customer Name	Receipt#	Sls#	Salesman Name	Units	Unit Price	Ext. Price
							Cost
							Ext. Cost
100	PC Distribution Inc.	129	Ernest Hernandez				
	4790	CX3000	15	\$1,800.00	\$27,000.00	\$1,530.00	\$22,950.00
	4796	PB3050	8	\$3,000.00	\$24,000.00	\$2,300.00	\$18,400.00
	5081	PX3050	5	\$3,000.00	\$15,000.00	\$2,350.00	\$11,750.00
	5060	MX3010	10	\$1,700.00	\$17,000.00	\$1,630.00	\$16,300.00
	5080	PB3040	15	\$2,250.00	\$33,750.00	\$2,175.00	\$32,625.00
Customer 100 Subtotal		53			\$116,750.00		\$102,025.00
Transaction Totals				349		\$760,225.00	\$641,670.00
Profit: 18.5%							


Figure 11-9 Breakpoint Subheadings and Subfootings on the Sample Report

## Setting Page Length

Page width can be up to 131 characters and is determined by the editing and locating functions. You can request any page length between one and 999 lines for the report. The minimum page length must be at least the number of heading, footing, breakpoint, and detail lines marked on the report format. The default page length is 66 lines. Set the page length to zero (0) to suppress all page breaks when the report is printed.

Most printers print six lines per inch vertically and ten characters per inch horizontally. This means that 66 lines is equal to 11 inches. Standard paper is 8-1/2 by 11 inches which translates to 85 columns wide by 66 lines long. Wide carriage printers can handle paper widths up to 14 inches so the widest possible report (131 characters) can be printed on a wide carriage printer. If your printer is constructed for 8-1/2 inch wide paper and you want to print reports wider than 85 columns, see "Setting Printer Control Variables" in this chapter.

To change the page length, select *Set* from the REPORTS menu. REPORTS responds with a screen similar to the following:

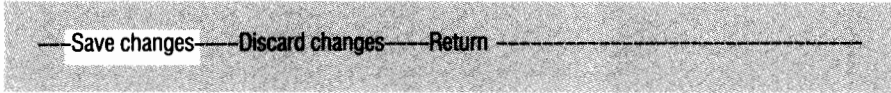


Current number of lines per page is 66. New number:

Enter the new page length at the *New number* prompt or press [ENTER] to leave it the same. REPORTS redisplay the REPORTS menu.

## Leaving Report Generation

When you are satisfied with your report format, select *Quit* from the REPORTS menu to save your report and exit to the R:base prompt. REPORTS displays the following screen:



—Save changes—Discard changes—Return—

- Select *Save changes* to store the new report or the report changes
- Select *Discard changes* to discard any changes made
- Select *Return* to return to the REPORTS menu

If you selected *Save changes*, you can now test your report. Follow the description in the "Printing the Report" section of this chapter. Return to the "Editing the Report" section for guidance in making any adjustments.

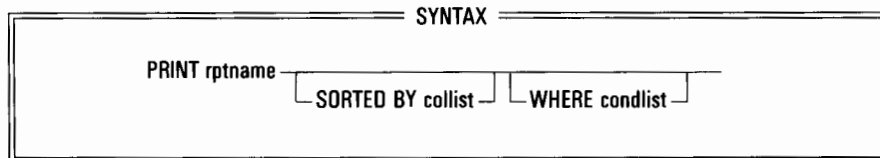
## PRINTING THE REPORT

You have three report output options:

- Output to the screen
- Output to the printer
- Output to a file



For all three options, you use the PRINT and OUTPUT commands. These commands are entered at an R> prompt, outside of REPORTS. The PRINT command syntax looks like this:



This command uses the report format specified in *rptname*. You can state the sort order of the report by adding the SORTED BY clause. Up to 10 table columns can be included in the sort list. For example, the *salesrpt* example report can be printed with the following command typed at the R> prompt:

```
PRINT salesrpt SORTED BY custid prodid
```

This first sorts the transactions in the *transx* file in order by customer then subsorts them by product number.

If you have set breakpoints in your report, you do not need to use the SORTED BY clause. R:base automatically sorts the table for reporting by the table columns defined as break columns. However, if your breakpoints are on variables, you must include the SORTED BY clause for any sorting you need. A SORTED BY clause in the PRINT command takes precedence over breakpoint column sorting.

In addition to sort order, you can limit the rows reported by adding a WHERE clause to the command as follows:

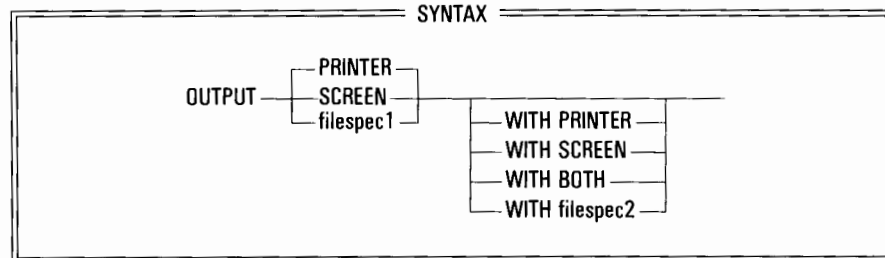
```
PRINT salesrpt WHERE tdate = 03/11/85
```

This command produces the daily transactions for March 11, 1985 only. This is the format used to produce the sample report.

If you do not remember the name of the report you want to print, type the LIST command at the R> prompt to list the available report formats:

#### LIST REPORTS

The OUTPUT command directs the report to a specified output device. You can also specify a secondary output device. The OUTPUT command looks like this:



### Output to the Computer Screen

First, make sure that your output device is set to SCREEN by entering the following command:

```
OUTPUT SCREEN
```

Then, use the PRINT command as described above to display the report on the screen. Remember that the report width and length are probably set for printing on a printer. Therefore, the report format as shown on the screen may differ from the printed report. Screen displays are always 80 characters wide, therefore longer lines will appear on two lines on the screen. The screen is the default output device in R:base.

### Output to the Printer

To print your report, you use the OUTPUT command and then the PRINT command. For the sample report, the command sequence looks like this:

```
OUTPUT PRINTER
PRINT salesrpt WHERE tdate = 03/11/85
OUTPUT SCREEN
```

This command sequence performs the following functions:

- Sets your output device to the printer.
- Prints the report on the printer. Transactions dated March 11, 1985 only are printed.
- Redirects output to the screen.

Use this command sequence to print the report named *salesrpt* in the *compuco* database. If your printer is only 80 columns, change the report to send a compressed print escape code to your printer. See "Setting Printer Control Variables" in this chapter for procedures. If your printer has an adjustable pitch setting, you can set it to 15 pitch instead of entering printer control codes in the report.

### PRINTING ON SINGLE SHEETS

To print on non-continuous (single sheet) forms, use the OUTPUT WITH SCREEN form of the OUTPUT command. For the sample report, you enter the following command sequence:

```
OUTPUT PRINTER WITH SCREEN
PRINT salesrpt WHERE tdate = 03/11/85
OUTPUT SCREEN
```

This command sequence performs the following functions:

- Sets your output device to the printer with simultaneous output to the screen. At the end of each page, R:base prompts you to press any key to continue. Load the next sheet of paper on the printer and then press any key.
- Prints the report on the printer. Transactions dated March 11, 1985 only are printed.
- Redirects output to the screen.

### Output to a File

To send your report to a file, you use the OUTPUT command with a file name. For the sample report, you enter the following command sequence:

```
OUTPUT d:sales.rpt
PRINT salesrpt WHERE tdate = 03/11/85
OUTPUT SCREEN
```

This command sequence performs the following functions:

- Specifies that the formatted report is to be stored in the file named *sales.rpt* located on drive *d:*. If the file does not exist, R:base creates it. If the file already exists, it will be overwritten.
- Sends the report to the specified file. Transactions dated March 11, 1985 only are sent to the file.
- Redirects output to the screen.



## DELETING REPORTS

All report formats are saved in a table called *REPORTS*. This table has two columns: *RNAME* and *RDATA*. The column *RNAME* always holds the report name and *RDATA* contains the report format information. If you have created a report or you have the *compuco* sample database open, type:

```
SELECT ALL FROM REPORTS
```

If you want to delete the report you can use the *DELETE ROWS* command with a *WHERE* clause. For the sample report, the command looks like this:

```
DELETE ROWS FROM REPORTS WHERE RNAME = salesrpt
```

## ADVANCED REPORT CONCEPTS

### Using Variables and Route Numbers

Although the *REPORTS* lookup feature (see “Defining Columns from Other Tables”) allows you to retrieve column data from other tables, in some situations it is faster to use the *SET POINTER* and *SET VARIABLE* commands to gather report data. For example, suppose you want to print a report on a single company’s sales using the transaction table. Using a lookup in *REPORTS* is slower than using a *SET POINTER* command and global variables within the report.

First, you enter a *SET POINTER* command to point to the individual company’s row in the customer table:

```
SET POINTER #1 route FOR custtbl WHERE custid = 1000
```

Next, you set global variables to hold the column values from the company’s row. The company table contains columns for credit limit and year-to-date sales that you want for each report line. You need to set variables similar to the following:

```
SET VARIABLE credlim TO credit IN #1  
SET VARIABLE ytd TO ytdsales IN #1
```

In your report variable definitions, you then set a report variable to the value of the global variable:

```
credlims = .credlim  
ytds     = .ytd
```

You then *Locate* the two report variables or use them to calculate other variables. This executes much faster than using a lookup to retrieve the two columns each time they are needed in the report.

## Making Mailing Labels

Mailing labels are easy to produce using REPORTS. In the sample database *compuco*, the *custlist* table contains all the columns necessary for producing mailing labels. Only two variables need to be defined to combine the first and last names and the city, state and zip code of the customer contact.

First, define the text variable *custname* using the columns *cfname* and *cname*. The variable expression looks like this:

```
1:TEXT      : custname  = cfname & cname
```

On the sample labels, combine the city, state, and zip code using the following variable definitions:

```
2:TEXT      : comma    = " , "
3:TEXT      : ccity     = city + comma
4:TEXT      : cstate    = state & czip
5:TEXT      : citystat   = ccity & cstate
```

If the city, state, and zip code were located individually, with a comma entered through the *Edit* option, the line would print like this:

```
Boston      , MA 02178
```

By combining the text columns as shown above, the *citystat* variable line is printed like this:

```
Boston, MA 02178
```

Next, use the *Locate* option to mark the positions of the company name, contact name, address, and combined city, state, and zip code as shown in the following screen. The column and variable names are shown in italics although they do not appear on the screen.

```
Name of column or variable:company      < 4, 40>      [F3] to list, [ESC] to exit
| D | S  company                        E
| D | S  custname                       E
| D | S  address                        E
| D | S  citystat                       E □
```

Next, use the *Mark* option to mark detail lines. Labels do not contain any heading, footing, or breakpoint lines. Each label used in the example is three inches, equivalent to 18 lines (6 lines per inch times a 3 inch label). Mark 18 lines as detail lines starting with the first line. This allows for the unused lines on the label.

Finally, use the *Set* option to set the number of lines to 0. This suppresses any page ejects (form feeds) so that the labels print continuously until all of the selected rows are printed.

When you are ready to print labels, align the print head of the printer on the exact spot where you want printing to begin. On the sample labels, the labels are preprinted with TO: in the middle of the label. Position the print head next to the word TO: on the label.

It is a good idea to print a practice label for alignment. To do this, limit the rows selected for printing with a WHERE clause in the PRINT command. Use the following command sequence to print a sample label for alignment:

```
OUTPUT PRINTER  
PRINT labels WHERE LIMIT = 1
```

The PRINT command can be repeated until you are satisfied with the alignment. When you are ready, use the PRINT command with any WHERE or SORTED BY clauses you need to limit and sort the information. For the sample labels, use the following command sequence:

```
OUTPUT PRINTER  
PRINT labels SORTED BY zip
```

This prints the labels in zip code order. A sample of the labels produced is shown in figure 11-10.



TO: Industrial Computers Inc.  
Jane Ferguson  
5200 Empire Way  
Denver, CO 80214

Figure 11-10 Sample Mailing Label

Be sure to issue an OUTPUT SCREEN command after printing the labels to redirect the output to the screen.

## Creating Form Letters

You can use REPORTS to create form letters by retrieving columns from other tables and using the text-wrap features.

First, create the form letter. You can use any word processor which creates ASCII text files, such as WordStar™ in non-document mode. Enter each paragraph as a single text string. Do not enter carriage returns at the end of each line. Begin and end each paragraph with a quotation mark (""). Each paragraph becomes a column in the form letter table. The paragraphs look something like this:

"This is the first paragraph in the form letter text. It becomes the first column in the form letter table."

"This is the second paragraph in the form letter text. It becomes the second column in the form letter table."

"This is the third and final paragraph although there can be as many paragraphs as you want."

Next, define a table to hold the form letter text. Each paragraph becomes an individual column in the table. The table looks something like this:

Table: formlet

columns

#Name	Type	Length	Key
1 para1	TEXT	200 characters	
2 para2	TEXT	200 characters	
3 para3	TEXT	200 characters	

Be sure to make the text lengths of each column long enough to hold the paragraph.

Load the form letter text into the table using the `LOAD . . . AS ASCII` command, for example:

```
LOAD formlet AS ASCII USING para1 para2 para3
```

Next, define the report format to print the form letter material. Use the same techniques described under "Making Mailing Labels" to format the name and address at the top of the letter. Define the name, address, salutation (Dear . . .) as breakpoint subheadings and the closing (Sincerely,) as a subfooting. The breakpoints are determined by changes in the customer identification number. Request a page eject following the page footing.

Use variables to format the city, state, and zip code; combine the first and last names of the customer contact; and define a variable for the date as shown below:

```
1:TEXT      :comma      = ","
2:TEXT      :ccity       = city + comma
3:TEXT      :cstate      = state & zip
4:TEXT      :citystat    = ccity & cstate
5:TEXT      :custname    = cfname & clname
6:DATE      :ldate       = #date
```

The text paragraphs are merged into the report with lookup variables. For the example, the lookup variables look like this:

```
7:TEXT      : parag1    = para1 IN formlet WHERE para1 exists
8:TEXT      : parag2    = para2 IN formlet WHERE para2 exists
9:TEXT      : parag3    = para3 IN formlet WHERE para3 exists
```

These three variables are located as text wrap variables with indentation. The lines are marked as detail lines. The completed report layout is shown on the screen below. The column and variable names are included in italics although they are not displayed on the screen.

```

                                     < 1, 1> [F3] to list, [ESC] to exit.
                                     S ldate E
| H1 | D
| H1 | S      company      E
| H1 | S      custname     E
| H1 | S      address      E
| H1 | S      citystat     E
| H1 |
| H1 | Dear Mr/Mrs S      clname      E
| H1 |
| D | S      I      parag1      W
| D |
| D | S      I      parag2      W
| D |
| D | S      I      parag3      W
| D |
| F1 |
| F1 |      Sincerely,
| F1 |
| F1 |
| F1 |      George Jones

```

Finally, the form letter is printed using the PRINT command.

## Duplicating a Report Format

Once you have perfected a report format, you may want to save some labor by duplicating it and then modifying it slightly for a different reporting function. You may also want to copy a report from one database to another. Procedures for both types of duplication follow.

### DUPLICATING A REPORT WITHIN A DATABASE

Before duplicating a report, back up the database. Open the database containing the report you want to duplicate. At the R:base R> prompt, use the following sequence of commands. *Temprel* is the temporary table, *oldrpt* is the existing report, and *newrpt* is the report you want to add.

```

PROJECT temprel FROM REPORTS USING ALL WHERE RNAME EQ oldrpt
CHANGE RNAME TO newrpt IN temprel WHERE RNAME EQ oldrpt
APPEND temprel TO REPORTS

```

This command sequence performs the following functions:

- The PROJECT command creates a table named *temprel* from the old report named *oldrpt* using all columns in the table.
- The CHANGE RNAME command changes the name of the report from *oldrpt* to *newrpt* in all rows of the temporary table *temprel* in the *RNAME* column.
- The APPEND command adds the new report to the *REPORTS* table. If you now request the report by its new name *newrpt*, R:base displays the report format you originally set up as *oldrpt*. Make whatever modifications you need. Then use the report. Remember, the table reported is the same.

After duplicating the report, use the REMOVE command to delete the temporary table *temprel*. To do this, at the R> prompt type:

```
REMOVE temprel
```

### **COPYING A REPORT TO A DIFFERENT DATABASE**

Open the database which holds the existing report. Use the following command sequence to duplicate the report.

```
OUTPUT d:temprpt.dat
UNLOAD DATA FROM REPORTS WHERE RNAME EQ oldrpt
OUTPUT SCREEN
OPEN newdb
INPUT d:temprpt.dat
```

This command sequence performs the following functions:

- The OUTPUT command specifies the file *temprpt.dat* residing on drive *d:* as the output file.
- The UNLOAD command writes out the rows (DATA only) from the *REPORTS* table containing the format of the *oldrpt*. The data is placed in the output file *temprpt.dat* on drive *d:*.
- The OUTPUT SCREEN command resets the output device to the screen.
- The OPEN command opens the database where you want the report format moved.
- The INPUT command loads the report format into the *REPORTS* table. If you use the TYPE command on the input file, you see the LOAD REPORTS command followed by the data to be loaded.

If you also want to change the report name while duplicating the report, use the methods described in “Duplicating a Report Within a Database” in this chapter.

A final tip: You can also change the table name in the report by using the **CHANGE** command as described above. However, when you attempt to print the report or change its format, R:base displays error messages for any column names which are incorrect. When you first attempt such modifications, it is a good idea to first back up your database.

## Decimal Point Alignment for Real Numbers

Real numbers are printed with significant digits of each number to the right of the decimal point. You can, however, align real numbers by specifying the exact position of the decimal point within the location for the real number. To do this, use *Edit* mode to enter a decimal point (.) where you want it to appear in the column of numbers. The *E* ending mark is considered to be one character position. Following are examples of what the location field looks like, the actual real numbers to be printed, and the resulting output format.

Location Mask	Original Number	Output Format
S . E	8.999	8.99
	422.9	422.90
	.999999	.99
S .E	8.999	9.0
	422.9	422.9
	.999999	1.0
S . E	8.999	8.999000
	422.9	422.900000
	.999999	.999999

R:base attempts to format the real numbers to the requested mask but, if the number is too large (scientific notation format) or if the accuracy of the number is compromised, it reverts to the original real number format.







## Data Transfers Contents

<b>How to Use This Chapter</b>	12-2
<b>Transferring R:base Tables to Other Systems</b>	12-3
Transfer Commands	12-3
UNLOAD	12-3
OUTPUT	12-5
Unloading R:base into Delimited ASCII Files	12-5
Unloading R:base into DIF Files	12-7
Unloading R:base into Multiplan SYLK Files	12-8
Unloading R:base into ASCII Fixed Field Files	12-9
<b>Transferring Files into R:base</b>	12-10
Preparing the Database for File Transfers	12-10
Transfer Commands	12-10
LOAD	12-10
INPUT	12-11
ENTER	12-11
Loading Delimited ASCII Files into R:base	12-12
Loading ASCII Delimited Records	12-13
Loading ASCII Data with Blanks in Text Strings	12-13
Loading ASCII Data with Null Values	12-14
Loading Fixed Field ASCII Files into R:base	12-15
Loading ASCII Fixed Field Files	12-15
Loading Records Longer Than 79 Characters	12-16
<b>Combining Use of the Transfer Commands</b>	12-18
Transferring Tables Between Databases	12-18
Transferring Data for Back Up	12-19
Embedding R:base Commands in an Input File	12-20
Using an Editor to Add Commands to an Input File	12-20
Using UNLOAD to Create an Input File	12-22
<b>Interfacing with Word Processing Packages</b>	12-22

## HOW TO USE THIS CHAPTER

This chapter describes how to transfer data between R:base and other systems using the UNLOAD, LOAD, OUTPUT, INPUT, and ENTER commands. You can also use these commands to transfer a table from one R:base database to another or to interface with word processing programs.

The first section explains how to use UNLOAD to convert R:base tables into files usable by mainframe systems, VisiCalc, Lotus 1-2-3, Multiplan, dBASE II, or other software using ASCII delimited or fixed field files, and DIF or SYLK file formats.

The second section describes how to use LOAD to convert ASCII files into R:base tables.

The third section contains instructions for combining use of transfer commands for transferring tables between databases, for back up and recovery, and embedding R:base commands in an input file.

The fourth section covers a method for using R:base tables with word processing mailing list utilities.

Table 12-1 lists file conversions you may want to perform and the recommended method of transfer.

Table 12-1 File Conversion Methods

From	To	Method
R:base	ASCII Delimited	UNLOAD AS ASCII
R:base	DIF File*	UNLOAD AS DIF
R:base	Multiplan SYLK†	UNLOAD AS MPLAN
R:base	ASCII Fixed Field	REPORTS and PRINT
R:base	R:base	LOAD and UNLOAD
ASCII Delimited	R:base	FileGateway or LOAD AS ASCII
ASCII Fixed Field	R:base	FileGateway or ENTER
dBASE II	R:base	FileGateway
DIF File*	R:base	FileGateway
Lotus 1-2-3	R:base	FileGateway
PFS:FILE	R:base	FileGateway
SYLK File†	R:base	FileGateway

\*VisiCalc, Lotus 1-2-3, or other program's DIF format

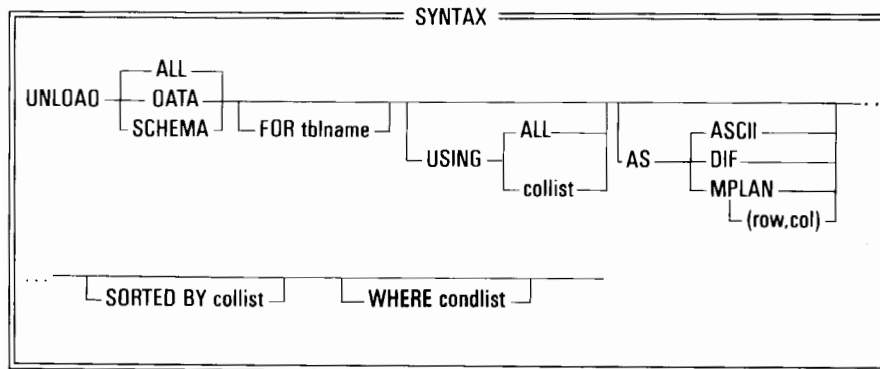
†Multiplan or other program's SYLK format

## TRANSFERRING R:BASE TABLES TO OTHER SYSTEMS

### Transfer Commands

#### UNLOAD

The R:base UNLOAD command is used to convert R:base database tables into formats usable by other systems. You have the option of conversion to ASCII delimited format, DIF format, or SYLK format. The syntax of the UNLOAD command is:



The UNLOAD command looks far more complicated than it really is. Table 12-2 explains each part of the command.

Table 12-2 The UNLOAD Command

Command Phrase	Meaning
UNLOAD ALL	Unloads both data and database structure.
DATA	Unloads data only.
SCHEMA	Unloads the database structure only.
	One of the above types must be entered.
FOR tblname	Unload the named table. If omitted, all tables in the open database are unloaded. If the WHERE or SORTED BY clauses are included, then <i>tblname</i> must be used.
USING collist	Unload the named columns (optional).
ALL	Unload all columns (optional).
AS ASCII	Unload the file to ASCII delimited format.
DIF	Unload the file to DIF format.
MPLAN	Unload the file to SYLK format
(row,column)	Indicates the Multiplan row and column.
SORTED BY collist	While unloading, sort the rows by the indicated columns (up to 10) (optional).
WHERE condlist	Unload only the rows meeting the conditions specified (optional). Cannot be used if the UNLOAD SCHEMA form is used.

The UNLOAD DATA . . . AS ASCII combination has a different effect than the UNLOAD DATA command without the AS ASCII clause. Without the AS ASCII clause, the file includes a few R:base commands which allow the unloaded data to be reloaded into a defined table using the INPUT command. With the AS ASCII clause, the output file contains data only in ASCII delimited format.

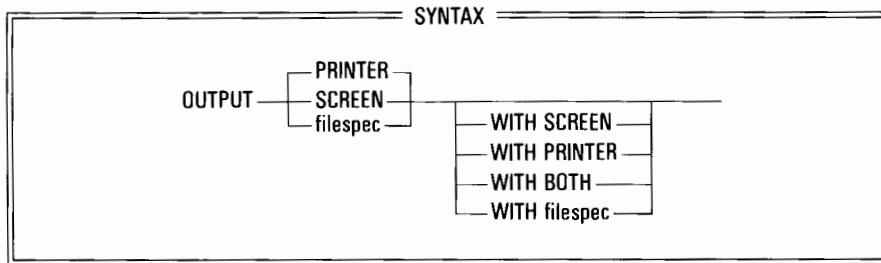
Additionally, the AS ASCII clause unloads each row as one continuous line. Without the AS ASCII clause, the data is unloaded as separate lines of 79 or fewer characters, with a plus sign (+) at the end of each line to indicate continuation to the next line.

When unloading ASCII data to a non-R:base system, be sure to include the AS ASCII clause to eliminate the R:base commands.

The UNLOAD SCHEMA form of the command is useful for checking user and owner passwords. If the user and owner passwords do not match, you cannot unload the structure. If the passwords match, you see what they are on the screen. Use the command without redirecting the output to the printer or a file.

## OUTPUT

The OUTPUT command allows you to indicate where the unloaded data is to be sent. The OUTPUT command syntax is:



In all examples used in this chapter, it is assumed that the output will be put into an external file named *filespec*. When the unloading process is complete, you must use the OUTPUT command a second time to close the file and send the output back to the terminal (SCREEN). The word TERMINAL may be used instead of SCREEN.

Notice that you have the option of simultaneously displaying and printing the data as it is being unloaded.

The OUTPUT and INPUT commands are not directly related. When you change the output device with OUTPUT, you do not use INPUT to reset the device. You use OUTPUT again and specify a different output device.

## Unloading R:base into Delimited ASCII Files

Unloading your R:base tables as ASCII delimited files is useful for

- Making back-up copies
- Keeping a historic record of daily data entries
- Transferring tables from one machine to another
- Interfacing with other software systems

To transfer R:base tables to ASCII delimited format, type either form of the UNLOAD command at the R> prompt:

UNLOAD DATA FOR tblname AS ASCII

or

UNLOAD ALL FOR tblname AS ASCII

It is sometimes best to unload the database structure (SCHEMA) into one file and the data into another instead of using the UNLOAD ALL option. This allows you to use the R:base editor RBEDIT to correct the structure file for such things as the drive indicator on the database name.

If you want to send the table to an R:base compatible database management system such as RIM, use the UNLOAD ALL form to output both data and table structure. If you want to send the table as a file for use with other software, use the UNLOAD DATA form combined with the appropriate AS clause to send only the data. The UNLOAD DATA form without an AS ASCII clause transfers the data portion of the file in ASCII delimited format but includes a few R:base commands as well. These commands appear to be R:base comments but are actually fully operational SET commands. Another form of the command is UNLOAD SCHEMA which is used to transfer the table structure alone.

You can also indicate which columns to unload with the USING clause, specify a sort sequence with the SORTED BY clause, and set conditions for the columns to be unloaded with a WHERE clause. The formats of the USING, SORTED BY and WHERE clauses are as follows:

```
USING collist
SORTED BY collist
WHERE condlist
```

The order specified in the USING clause is the order used to unload the data into the table. You can specify USING ALL to unload all columns in the same order as in the table.

Following is the basic sequence of commands used to carry out the unloading process. You can add any of the optional clauses specified above.

```
OPEN d:compuco
OUTPUT d:adrlist.dat
UNLOAD DATA FOR custlist AS ASCII
OUTPUT SCREEN
```

This command sequence performs the following functions:

- Opens the database *compuco* located on drive *d:* from which the table is to be transferred
- Specifies file *adrlist.dat* on drive *d:* as the output file name
- Outputs the data portion of the table *custlist* to the output file *adrlist.dat* on drive *d:*
- Redirects output to the screen

R:base assumes ASCII delimited format if no AS clause is included. However, R:base inserts additional R:base commands into the file which allow you to load the unloaded file back into R:base using the INPUT command. You can omit the AS ASCII clause if you are unloading a table to another database into a table of the same name or creating a back up copy. See “Embedding R:base Commands in the Input File” in this chapter for an example of an ASCII delimited file unloaded without the AS ASCII clause.

## Unloading R:base into DIF Files

To transfer R:base tables to systems which can accept DIF format files, use this format of the R:base UNLOAD command:

```
UNLOAD DATA FOR tblname AS DIF
```

You can also specify which columns to unload with a USING clause, specify a sort sequence with a SORTED BY clause, and set conditions for the rows with a WHERE clause:

```
USING collist  
SORTED BY colname  
WHERE condlist
```

The order specified in the USING clause is the order used to unload the data into the table. You can specify USING ALL to unload all columns in the same order as in the table.

Following is the basic sequence of commands used for transferring an R:base table to DIF format:

```
OPEN d:compuco  
OUTPUT d:custlist.dif  
UNLOAD DATA FOR custlist AS DIF  
OUTPUT SCREEN
```

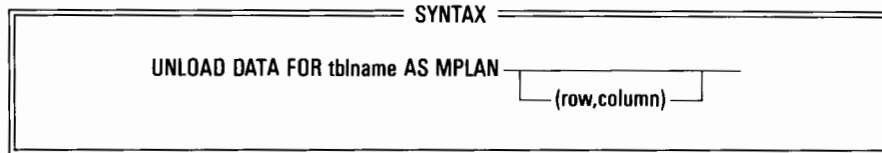
This command sequence performs the following functions:

- Opens the database *compuco* located on drive *d:* from which the table is to be transferred
- Specifies that the output should be sent to the file named *custlist.dif* located on drive *d:*
- Transfers the data in the *custlist* table into DIF format and places it into the output file *custlist.dif* on drive *d:*
- Redirects output to the screen



## Unloading R:base into Multiplan SYLK Files

To transfer R:base tables to systems which can accept SYLK format files, use this form of the UNLOAD command:



You have the option of specifying the starting spreadsheet row and column for the data. This allows you to place the unloaded data anywhere in the Multiplan worksheet when you reload the SYLK file into Multiplan.

You can also specify which columns to unload with a USING clause, specify a sort sequence with a SORTED BY clause, and set conditions for the rows with a WHERE clause:

```

USING collist
SORTED BY collist
WHERE condlist

```

Be sure to include the USING clause if you need to reorder the columns to match the Multiplan spreadsheet format.

Following is the basic sequence of commands used to transfer an R:base table to SYLK format:

```

OPEN d:compuco
OUTPUT d:custlist.syl
UNLOAD DATA FOR custlist AS MPLAN
OUTPUT SCREEN

```

This command sequence performs the following functions:

- Opens the database *compuco* located on drive *d:* from which the table is to be transferred
- Directs the output to the file named *custlist.syl* located on drive *d:*
- Transfers the data in the *custlist* table into SYLK format and places it into the output file *custlist.syl* on drive *d:*
- Redirects output to the screen

You can also specify the starting row and column, or specify the column order with the USING clause.

Once you have created the SYLK file, you then use the Multiplan program to load the file into Multiplan format from SYLK format.

## Unloading R:base into ASCII Fixed Field Files

Many products, including some word processing systems, can use ASCII fixed field files where they cannot use ASCII delimited files because of slight format differences such as the delimiter character.

Another advantage to using ASCII fixed field files is that you can easily transfer selected columns in any order you want.

In R:base, you use **REPORTS** to create and format a report and the **PRINT** command to create an ASCII fixed field file. The **UNLOAD** command is not used for this process.

First, you create a report format which matches the maximum lengths of the fields you want to transfer (see chapter 11). When you are creating the report format:

- Do not enter any headings or footings
- Locate all the fields on a single detail line with no spaces between fields
- Do not use the dollar mask format
- Set the page length to zero (0)

When the report format is prepared, use the following command sequence to create the ASCII fixed field file:

```
OUTPUT d:ascii.dat  
PRINT rptname  
OUTPUT SCREEN
```

This command sequence performs the following functions:

- Directs the output to the file *ascii.dat* on drive *d:*
- Sends the report output to the file in ASCII fixed field format
- Redirects the output to the screen

## TRANSFERRING FILES INTO R:BASE

### Preparing the Database for File Transfers

With the R:base LOAD, INPUT, and ENTER commands, you can transfer ASCII files into your R:base database.

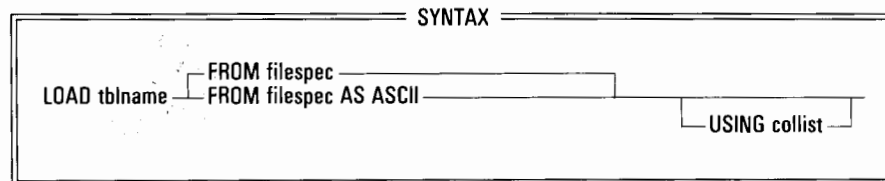
Before you transfer the files, however, you must prepare a database table for the data. This is accomplished by any of the following methods:

- Using the R:base DEFINE, COLUMNS and TABLES commands as described in chapter 5
- Embedding the column and table definitions in a command file along with the data to be transferred as described in “Embedding R:base Commands in an Input File” in this chapter
- Using the Application EXPRESS system as described in chapter 14

### Transfer Commands

#### LOAD

The LOAD command transfers data from other files into predefined R:base tables. The syntax for the LOAD command is as follows:



*Filespec* is a file designation in the form *d:\path\filename.ext*, where *d:* is the optional drive letter, *path* is the subdirectory, *filename* is the 1 to 8 character file name, and *.ext* is the 1 to 3 character file extension. Remember to include the drive letter in the file name if the file is not on the default drive. For example, *b:\mydir\example.dat*.

Before using the LOAD command you need to define the format of the table and columns into which the data is to be loaded. Use the AS ASCII clause for ASCII delimited files.

If you do not include an AS clause, R:base assumes that all rows are less than 79 characters, and that longer rows are continued with plus signs. If an ASCII delimited file has records longer than 79 characters continued on a single line, use the AS ASCII option.

You can add a USING clause when there is not a one-to-one correspondence between the columns defined in the table and the fields in the input file.

The LOAD command is also used to enter data from the keyboard. See chapter 7 for more information.

## INPUT

The INPUT command reads and executes R:base commands entered in an input file. The database, table, and columns are defined in the input file. The syntax of the INPUT command is:

<p style="text-align: center;">SYNTAX</p> <p style="text-align: center;">INPUT filespec</p>
---

When the file specified in the *filespec* is loaded, use the INPUT command a second time to reset the input device:

INPUT KEYBOARD

or

INPUT TERMINAL



Also see information on the RUN command.

## ENTER

The ENTER command may be used to enter ASCII fixed field data from an input file in the format designated by a data entry form, *formname* in the example below.

<p style="text-align: center;">SYNTAX</p> <p style="text-align: center;">ENTER formname FROM filespec</p>
---

The data in the file specified by *filespec* must be data in ASCII fixed field format. See "Loading Fixed Field ASCII Files into R:base" in this chapter.

The ENTER command is used with a different syntax to enter data from the keyboard. See chapter 7.

## Loading Delimited ASCII Files into R:base

ASCII delimited format is the most common format for transferring data to your R:base database from other databases or from files created by application systems. You can transfer ASCII delimited files into R:base using the `LOAD. . .AS ASCII` command.

In an ASCII delimited file, fields within each record are separated by a unique delimiter character and each record ends with a carriage return/line feed. You can use the `TYPE` command to display the first 79 characters of each line of an ASCII delimited file. It should look similar to this file which is named *adrlist.dat*:

```
"PC Distribution Inc.",Sarah,James,"3200 Westminster Way",Boston,MA,02178
"Industrial Computers Inc.",Jane,Ferguson,"5200 Empire Way",Denver,CO,80214
"Computer Mountain Inc.",Bill,Stevenson,"14792 15th Ave. E.",Denver,CO,80214
```

R:base recognizes one or more consecutive blanks or commas as the delimiter characters in the input file. In addition, text fields which contain blanks or commas must be set off with quotation marks—either single (') or double ("). Otherwise, R:base assumes them to be delimiters. If the delimiter character is a single quote ('), you must use the `SET QUOTES` command to change to the single quote format. Before attempting to load the data, type at the `R>` prompt:

```
SET QUOTES='
```

Be sure to set it back to double quotes when you are finished loading.

Many systems can create ASCII delimited files which can be loaded into R:base. For example, dBASE II and dBASE III files can be copied to delimited format. These programs use single quotes as delimiter characters so you will need to tell R:base to use single quotes by using the `SET QUOTES` command as shown above.

This section explains several variations on loading ASCII delimited files:

- Loading simple ASCII delimited records
- Loading data with blanks contained in text strings
- Loading data with null values

**LOADING ASCII DELIMITED RECORDS**

If your ASCII delimited file has no special differences such as embedded blanks in text fields or null values then the following form of the LOAD command is used:

```
LOAD tblname FROM file.dat
```

If you do not include the AS ASCII clause, R:base assumes the file is ASCII delimited format with lines either less than 79 characters long or continued with plus signs. Be sure to include the AS ASCII clause if your file records are longer than 79 characters.

If the response to the LOAD command is a +> prompt, your ASCII file is missing either a matching quotation mark or a right parenthesis. Enter either a quotation mark (") or a right parenthesis ( ) ) depending on which character you believe may be missing.

**LOADING ASCII DATA WITH BLANKS IN TEXT STRINGS**

If your ASCII input file has text strings containing blanks which are not surrounded with quotation marks and the delimiter character is not a blank, you can load the file by using the SET BLANK command as follows:

```
SET BLANK=#
LOAD#tblname#FROM#file.dat
SET#BLANK=
```

This command sequence performs the following functions:

- Changes the BLANK character to # (you can use any character which is not included in the file)
- Loads the table *tblname* with data from *file.dat*
- Changes the BLANK character back to a real blank

Note that you need to use the new BLANK character # when typing the commands following the SET BLANK command. The effect is that the real blanks are not recognized as such while the file is loaded, thereby allowing the text strings which are not surrounded with quotation marks.

To load the input file into the *custlist* table in the *compuco* database, type the following sequence of commands at the R> prompt:

```
OPEN d:compuco
LOAD custlist FROM d:adrlist.dat AS ASCII
```

This command sequence performs the following functions:

- Opens the R:base database named *compuco* residing on drive *d:*
- Reads the input file *adrlist.dat* residing on drive *d:* and loads the data into the table named *custlist*

**LOADING ASCII DATA WITH NULL VALUES**

Records in an ASCII file cannot be loaded if they include null values, for example:

```
"Industrial Concepts Inc.",,,"5602 Silverdale Way",Livermore,CA,94550
```

When null values are contained in a record, the file must be edited before loading, filling all null fields with the symbol -0- (dash zero dash). For example:

```
"Industrial Concepts Inc.",-0,-0-,"5602 Silverdale Way",Livermore,CA,94550
```

You can avoid having to edit each record by using the FileGateway (see chapter 13).

You can load records with null values at the end of the line in two ways, described below.

If the ASCII file does not contain all of the data you want entered into the table, you can define additional columns in the table. Then, when you load the ASCII file, enter a USING clause with the LOAD command. For example, you can define columns *phone* and *comments* for the *custlist* table even if the input file does not have data for those two fields. To load the data from the *adrlist.dat* file, type at the R> prompt:

```
LOAD custlist FROM d:adrlist.dat AS ASCII +  
USING company fname lname address city state zip
```

The *phone* and *comments* columns are loaded into R:base with a null value represented by the symbol -0-.

An alternate method is to use the FILL command to tell R:base to fill columns at the end of rows with null characters. To do this, you must enter the FILL command on the same line as the LOAD command like this:

```
LOAD custlist FROM d:adrlist.dat ; FILL
```

This has the same effect as the LOAD command entered with the USING clause. Remember to include the USING clause, however, if the input data is in a different order than the columns are defined in the table.

If you want to load a file with trailing null fields which was previously unloaded with R:base definition commands, you must edit the input file to add the FILL command following the LOAD command line. See "Using an Editor to Add Commands to an Input File" in this chapter.

## Loading Fixed Field ASCII Files into R:base

You can transfer ASCII fixed field files into R:base using the ENTER command. Fixed field ASCII files (sometimes called formatted files) are characterized by a columnar format where matching fields in each record are all the same length. An ASCII fixed field file might appear like this:

PC Distribution Inc.	Sarah James	3200 Westminster Way	Boston	MA02178
Industrial Computers Inc.	Jane Ferguson	5200 Empire Way	Denver	CO 80214
Computer Mountain Inc.	Bill Stevenson	14792 15th Ave. E.	Denver	CO 80214

Fixed field ASCII files must have the following characteristics to be transferred into R:base:

- The end of each record must be marked by return and line feed characters.
- The database, tables, and columns must be predefined in R:base.
- A formatted data entry form must be created in R:base. Fields in the input file need not be in the same sequence as the columns in the database table since the data entry form takes care of ordering. For example, if the input fields are *name*, *address*, and *city*, your table can have the columns in the order *address*, *city*, and *name* as long as you create the form with the fields in the same order as the input file.

### LOADING ASCII FIXED FIELD FILES

Define the table and columns to hold the transferred ASCII data, as described in chapter 6.

Next, create a form using the FORMS command. See chapter 7 for assistance. The purpose of the form is to provide a mask of the data which is in the ASCII fixed field file. The form must match the length of each data field in the input file. You do not enter labels on the form, just the start and end markers for the data fields. The form for the example data shown above might look like this:

```
E(dit form),L(ocate columns),Q(uit):L
S          ES  ES      ES          ES      ESES      E
```

The start and end marks should define the lengths of each field you want to transfer from the ASCII file into the *custlist* table.

Once the form is prepared, use the ENTER command sequence to transfer your ASCII fixed field file to R:base. The command syntax is:

```
ENTER formname FROM filespec
```

R:base takes data from the input file specified in the *filespec* and enters it in the table as defined in the data entry form *formname*.



**LOADING RECORDS LONGER THAN 79 CHARACTERS**

If the records in your input file are longer than 79 characters (the width limit of the form), you need to split each record into segments of 79 characters or less by inserting return and line feed characters between fields in the input records. When you create the form, use multiple lines to mark the fields. For example, the following record is longer than 79 characters:

10001PC Distribution Inc.	Sarah	James	3200 Westminster Way
Boston	MA02178101-999-2929		

You need to split the record after the street by inserting a carriage return and line feed. You can do this with a BASIC program which reads the ASCII fixed field file and generates a new ASCII file with the additional characters embedded in it. Figure 12-1 shows a BASIC program used to split ASCII fixed field records.

```

10 CT% = 1
20 OPEN "file.dat" AS #1 LEN = 102
30 OPEN "split.dat" AS #2 LEN = 104
40 FIELD #1,5 AS A$,25 AS B$,10 AS C$,10 AS D$,20 AS E$,10 AS F$:2 AS G$:
   5 AS H$:12 AS I$:2 AS J$
50 FIELD #2,5 AS AA$,25 AS BB$,10 AS CC$,10 AS DD$,20 AS EE$,2 AS LF$:
   10 AS FF$,2 AS GG$:5 AS HH$:12 AS II$:2 AS JJ$
60 GET #1,CT%
70 IF EOF(1) THEN GOTO 130
80 LSET AA$ = A$:LSET BB$ = B$:LSET CC$ = C$:LSET DD$ = D$:LSET EE$ = E$
85 LSET LF$=CHR$(13)+CHR$(10):LSET FF$ = F$:LSET GG$ = G$:LSET HH$ = H$
90 LSET II$ = I$:LSET JJ$ = J$
100 PUT #2,CT%
110 CT% = CT% + 1
120 GOTO 60
130 CLOSE #1
140 CLOSE #2
150 END

```

Figure 12-1 BASIC Program for Splitting Records

This program performs the following functions:

- Line 10 defines a counter to read the file records (CT%)
- Lines 20 and 30 open the file to be split and a new file to hold the split data
- Line 40 defines the input field lengths in *file.dat* including the ending carriage return and line feed
- Line 50 defines the input field lengths in *split.dat* and includes an additional carriage return and line feed in the middle of the file (LF\$)
- Lines 60, 80, 85, and 90 read each record from *file.dat* and transfers the data to *split.dat* along with the extra carriage return and line feed
- Line 70 checks for the end of the file (EOF) and, if the end of the file is reached, passes control to line 130
- Line 110 adds one to the counter (CT%)
- Line 120 returns control to line 60 to read another record
- Lines 130 and 140 close both files when all records are processed

Create a form that looks like this:

```

E(edit form),L(ocate columns),Q(uit):L
S ES ES ES ES E
S ESES ES E

```

When the file is loaded with the ENTER command, the entire record is brought into the table as a single row even though the original record was split.

## COMBINING USE OF THE TRANSFER COMMANDS

### Transferring Tables Between Databases

You can transfer a table from one database to another by using a combination of the LOAD and UNLOAD commands. This is useful when two or more operators are entering data at the same time on different machines and you want to combine their entries at the end of the day.

Also, you may want to transfer information from a table in one database to another database. For example, the production department enters transaction data during the month and the sales department wants a copy of the transaction data at the end of each month.

The following sequence of commands may be used to transfer a table from one database to another. For this example, the two tables must have the same name.

```
OPEN dbname1
OUTPUT d:temp.dat
UNLOAD DATA FOR tblname
OUTPUT SCREEN
OPEN d:dbname2
INPUT d:temp.dat
OUTPUT SCREEN
```

This command sequence performs the following functions:

- Opens the database containing the table being transferred
- Designates the file *temp.dat* on drive *d:* as the output device
- Copies the data from *tblname* into the output file
- Resets the output device to the screen
- Opens the database on drive *d:* to which the table is being transferred
- Loads the data from the input file *temp.dat* on drive *d:*
- Redirects output to the screen.

If the table in the second database has not been defined, the UNLOAD command should look like this:

```
UNLOAD ALL FOR tblname
```

This command includes the necessary R:base definition commands to define a new table. If the new table is to have a different table name or reside in a different database, you must modify some of the R:base commands contained in the output file. If the table is small enough, you may be able to use the R:base editor, RBEDIT, to make the changes. If the table is too large for RBEDIT, use another editor or a word processor to change the drive designation, database, or table names.

## Transferring Data for Back Up

ASCII delimited files make a good storage medium for back up copies of your database tables. At the end of each day's work, you unload the tables which were updated during the day. These tables should have a today's date column which is filled in when the data is unloaded. The date column is cleared at the beginning of the day by typing

```
CHANGE datecol TO " " IN tblname
```

Then, to enter today's date just before unloading the table, type:

```
CHANGE datecol TO mm/dd/yy IN tblname WHERE datecol FAILS
```

where *datecol* is the column to hold today's date, *mm/dd/yy* is the date entered in the date format you are using, and *tblname* is the table you want to unload.

Next, unload the data into a file by typing:

```
OUTPUT d:mmddyy.dat
UNLOAD DATA FOR tblname AS ASCII
OUTPUT SCREEN
```

This command sequence performs the following functions:

- Designates the file *mmddyy.dat* (using today's date) located on drive *d:* as the file to send the output. For example, *031585.dat* is the file name for the March 15, 1985 back up file. If you want to unload more than one table to a back up copy, add an identifying number or letter for each file. For example *031585A.dat*, *031585B.dat*, and so on.
- Unloads the data only without any R:base commands into the output file in ASCII delimited format.
- Redirects output to the screen.

The AS ASCII clause is included to make sure that:

- Only data is sent to the file
- All rows are output as single continuous lines
- You can load the data into any table name
- You can transfer the data to other software packages

The entire process can be automated by using a command file to prompt for the date which can then be used to both change the date in the table and to name the output file with the current date. See chapter 15 for information on using command files to automate processing.

You can, of course, back up the entire database by using the COPY command at the DOS level to copy all three database files to a floppy disk.

## Embedding R:base Commands in an Input File

R:base commands can be included in an input file in either of two ways:

- Using an editor to add R:base commands to an ASCII input file
- Using the UNLOAD ALL command to create an input file

### USING AN EDITOR TO ADD COMMANDS TO AN INPUT FILE

If you regularly load data from an external file, you can put the R:base LOAD command in the input file instead of typing it at the R> prompt each time. This is especially useful if you always include one or more of the LOAD command clauses.

If you add a LOAD command to an ASCII input file, it looks similar to the sample screen shown here.

```
LOAD custlist USING company fname lname address city state zip
"PC Distribution Inc.",Sarah,James,"3200 Westminster Way",Boston,MA,02178
"Industrial Computers Inc.",Jane,Ferguson,"5200 Empire Way",Denver,CO,80214
"Computer Mountain Inc.",Bill,Stevenson,"14792 15th Ave. E.",Denver,CO,80214
"Industrial Concepts Inc.",-0,-0,"5602 Silverdale Way",Livermore,CA,94550
END
```

Since this form of the LOAD command does not indicate an external file (FROM. . .), R:base assumes that data follows the LOAD command and is in ASCII delimited format like data entered at the keyboard.

First, open the database and, if needed, define the columns and tables. Then, load the data by typing the INPUT command at the R> prompt:

```
INPUT d:adrlist.dat
```

R:base accepts the commands and data from the file named *adrlist.dat* and loads the data in the file into the specified table.

When you use the INPUT command to load data, R:base expects to find commands as well as data in the input file. When R:base reads the word LOAD, it recognizes its own command and proceeds to execute it just as if it were typed at the keyboard. R:base also has a RUN command which is used to execute command files. See chapter 15 for information on command files.

For example, you can create an input file that looks like this:

```
DEFINE d:compuco
COLUMNS
company TEXT 40
fname TEXT 20
lname TEXT 20
address TEXT 25
city TEXT 20
state TEXT 2
zip TEXT 5
TABLES
custlist WITH company fname lname address city state zip
END
LOAD custlist
"PC Distribution Inc.",Sarah,James,"3200 Westminster Way",Boston,MA,02178
"Industrial Computers Inc.",Jane,Ferguson,"200 Empire Way",Denver,CO,80214
"Computer Mountain Inc.",Bill,Stevenson,"14792 15th Ave. E.",Denver,CO,80214
"Industrial Concepts Inc.", -0-, -0-,"5602 Silverdale Way",Livermore,CA,94550
END
```

This input file contains all of the commands needed to define a table and its columns and to load the data into the defined table. If the table is already defined, the DEFINE, COLUMNS, and TABLES sections of the input file need not be included. To bring the data into R:base, type the following command at the R> prompt:

```
INPUT adrlst.dat
```

You can use the R:base editor RBEDIT to create your input file framework and enter the data with the commands as long as the file is not too large. Be sure to end both the definition and data portions of the input file with the END command.

**USING UNLOAD TO CREATE AN INPUT FILE**

The UNLOAD ALL command creates an input file similar to the one shown above. If you want to transfer a table from one database to another which does not already have defined columns and a table for the data, you can use the UNLOAD SCHEMA command to create an input file to hold the database structure which can then be loaded with the INPUT command. First, enter *OUTPUT filespec* to define the input file. Then enter the UNLOAD ALL command.

You must edit the input file to change the name of the database on the first line or the database names must be the same. When you are ready to load the table into a different database, open the database to which you want to transfer the table, then execute the INPUT command.

If you need to transfer the data as well, use the UNLOAD DATA form of the command and use the INPUT command to load the data after loading the database structure.

**INTERFACING WITH WORD PROCESSING PACKAGES**

If your word processor can accept ASCII format files, you can use data from R:base tables to produce word processed form letters.

The following example shows how to transfer an R:base table to a WordStar non-document file that can be used with MailMerge. Check your word processing manual for input file requirements if you are using other word processing systems. In this example, the name and address from the table *custlist* are used. The data from the *custlist* table is transferred to MailMerge for use in addressing a customized letter.

Following is the basic sequence of commands used to complete the unloading process:

```
OPEN d:compuco
OUTPUT d:adrlist.dat WITH SCREEN
UNLOAD DATA FOR custlist AS ASCII USING company fname lname +
address city state zip
OUTPUT SCREEN
```

This command sequence performs the following functions:

- Opens the database *compuco* located on drive *d:* from which the table is to be transferred.
- Specifies file *adrlist.dat* on drive *d:* as the output file name. The phrase WITH SCREEN tells R:base to display the data on the screen while the unloading is taking place. You can also specify the columns to write to the file with a USING clause.
- Writes the selected columns of the table *custlist* to the output file *adrlist.dat* on drive *d:* in ASCII delimited format. You can include any of the optional clauses such as SORTED BY or WHERE in the UNLOAD command. Notice that the command line is continued by placing a plus sign (+) at the end of the first line.
- Redirects output to the screen.

You can then use the *adrlist.dat* file as a MailMerge data file. Use the WordStar document mode to set up your form letter with WordStar dot commands as shown in figure 12-2.

Use the M (MailMerge) option on the WordStar menu to print your letter.

```
.op
.df adrlist.dat
.rv company,frname,laname,address,city,state,zip
&frname& &laname&
&company&
&address&
&city&, &state& &zip&
Dear Mr./Ms. &laname&

(body of letter)

.pa
```

Figure 12-2 Setting Up an R:base Table in Wordstar to Use with MailMerge



Consult the user's manual for your word processor for information on how it manages form letters. You can make some adjustments by using the SET command before unloading the table to the file. For example, if your word processor requires single quotation marks (') rather than double quotation marks ("), you can use the SET QUOTE command to change them.

Your word processor may require file input in ASCII fixed field format rather than ASCII delimited format. See "Unloading R:base into ASCII Fixed Field Files" in this chapter for information.

To find out if your word processor is compatible with either form of ASCII file unloaded from R:base, try to use the unloaded file with your word processor. If you can successfully access the file and read the contents of the screen, you can then edit it to conform to the form letter requirements of your word processor.

---

## The FileGateway Contents

<b>How to Use This Chapter</b>	13-3
<b>FileGateway Overview</b>	13-3
How the FileGateway Works	13-3
Selecting the File to be Converted	13-4
Converting Data	13-5
Correcting Exceptions	13-6
Definition of Terms	13-7
Data Types and Formats	13-8
Orientation of Spreadsheets	13-9
Spreadsheet Rows as Database Rows	13-9
Spreadsheet Columns as Database Rows	13-10
Total and Calculation Fields	13-11
Specifying Spreadsheet Boundaries	13-11
Sample Conversion File	13-11
<b>General Conversion Procedures</b>	13-13
Help Menu	13-13
General File Conversion Procedures	13-14
Step 1: Start the FileGateway	13-14
Step 2: Select File Type	13-15
Step 3: Enter the Original File Name	13-15
Step 4: Enter the Database Name	13-16
Step 5: Enter Name of the Converted Table	13-16
Choosing an Existing Table	13-17
Step 6: Make Room on the Disk if Necessary	13-18
Step 7: Set Specific Parameters if Necessary	13-18
Step 8: Edit Field Characteristics	13-19
Step 9: Load the Data	13-20
Step 10: Correct and Load Exceptions	13-22
Correcting, Loading, or Deleting Individual Records	13-24
Loading or Discarding All Records	13-24
Exception Handling Options	13-25
Leaving the Program	13-26

<b>Specific Conversion Procedures</b>	13-27
VisiCalc Worksheets in DIF File Format	13-27
What You Need to Know Before Conversion	13-27
Converting a VisiCalc Worksheet to DIF Format	13-28
Converting a DIF File to R:base Format	13-29
Multiplan SYLK Files	13-30
What You Need to Know Before Conversion	13-30
Row Orientation	13-30
Column Orientation	13-31
File and Record Length	13-32
Linking Fields	13-32
File Conversion Editor	13-32
Exception Handling Editor	13-32
Converting a Multiplan Worksheet to SYLK Format	13-32
Converting a SYLK File to R:base Format	13-33
ASCII Files with Fixed Fields	13-35
What You Need to Know Before Conversion	13-35
Converting an ASCII Fixed Field File to R:base Format	13-36
ASCII Files with Delimiters	13-39
What You Need to Know Before Conversion	13-39
Converting an ASCII Delimited File to R:base Format	13-39
Lotus 1-2-3 Files	13-41
What You Need to Know Before Conversion	13-41
Worksheet and Record Length	13-42
Linking Fields	13-43
File Conversion Editor	13-43
Exception Handling Editor	13-43
Converting a Lotus Worksheet File to R:base Format	13-43
dBASE II Files	13-46
What You Need to Know Before Conversion	13-46
Converting a dBASE II File to R:base Format	13-49
PFS:FILE Files	13-51
What You Need to Know Before Conversion	13-51
Linking Files with Common Field Names	13-52
Field Names	13-52
File Conversion Editor	13-52
Exception Handling Editor	13-53
Converting a PFS:FILE File to R:base Format	13-53

## HOW TO USE THIS CHAPTER

The FileGateway is a powerful tool which you can use to convert files from a wide variety of sources and combine them in one multi-table database. It is not simply a translator.

This chapter describes how to use R:base with data from files created by dBASE II, Lotus 1-2-3, and PFS:FILE as well as with DIF files from VisiCalc, SYLK files from Multiplan, and ASCII files from various popular programs or from a mainframe computer. The R:base LOAD command can be used to load ASCII delimited format files into an R:base database. However, the FileGateway provides the ability to specify column names and data types at the time of conversion, making the conversion faster and more efficient when building new tables.

The chapter begins with an overview of how the FileGateway works, suggestions for converting spreadsheet files, and a description of the sample data files included on the FileGateway disk. After reading the general conversion procedures, you can use the sample files for practice, following the specific procedures listed for converting DIF, SYLK, ASCII, Lotus 1-2-3, dBASE II, and PFS:FILE files.

## FILEGATEWAY OVERVIEW

### How the FileGateway Works

If Production uses dBASE II, Marketing uses Multiplan, and Administration uses PFS:FILE, the FileGateway lets you combine these files into one database. A file-server in a local area network is a convenient way to share files to be converted with the FileGateway.

The FileGateway does not alter your original files. It reformats a copy of these files into an R:base database format and copies them to a floppy or hard disk. You can still use the original files with the software that created them. In the instructions that follow, the translating and copying process is called *converting*. The unconverted file is referred to as the *original file* and the converted file as an R:base *table*.

The files you convert can be used with or enhanced by various Microrim products. For example, you can specify keyed columns with R:base in order to operate more efficiently and therefore faster.

In addition, you can tailor your database by reorganizing converted tables, defining new tables with various data combinations from the converted tables, or even removing extraneous data from the converted tables.



The FileGateway can convert files with records up to 1600 characters long as long as at least 70 of the characters are expendable. Once converted, the record length is 1530 characters since the FileGateway drops extraneous blanks, quotation marks, and delimiters. An R:base database can contain up to 400 columns in up to 40 tables. These limits are sufficient for most databases.

Figures 13-1 through 13-3 show the steps the FileGateway follows when converting files.

### SELECTING THE FILE TO BE CONVERTED

After you start the FileGateway, you are asked to identify the kind and name of the file to be converted and to name the database that will hold the converted tables.

You can include up to 40 separate tables in one database even if they are from different software products. If the original files are in the same format, you can add information from one file to the end of another or replace the data in the database table. Figure 13-1 illustrates the conversion process.

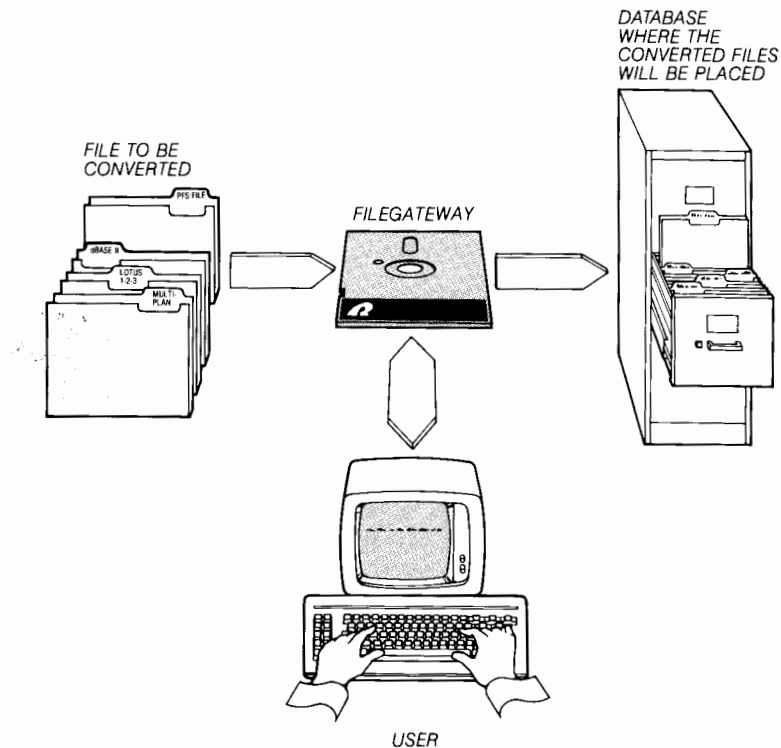


Figure 13-1 Selecting the File to be Converted

## CONVERTING DATA

After entering the file and database names, the FileGateway begins to convert your file. For most files, the FileGateway has enough information to convert the files to the R:base database format immediately. You are presented with a sample of the converted table to make changes to column names, data types, and data lengths, if needed.

Figure 13-2 shows how the FileGateway converts your file into an R:base format and, during the loading process, stores any records that it cannot load into an exception file.

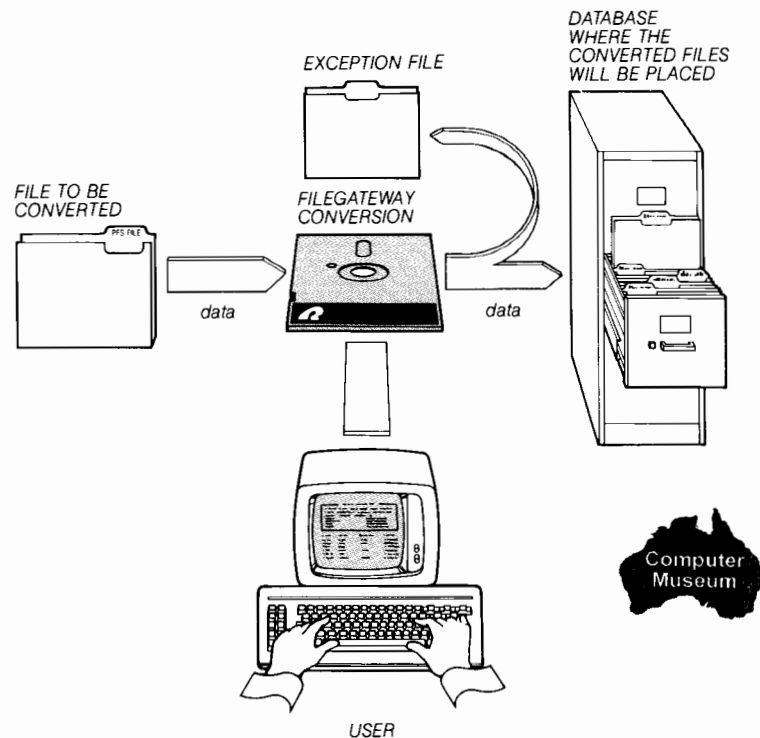


Figure 13-2 Converting and Storing Exceptions

### **CORRECTING EXCEPTIONS**

You can then correct and load the exceptions or delete them as shown in figure 13-3.

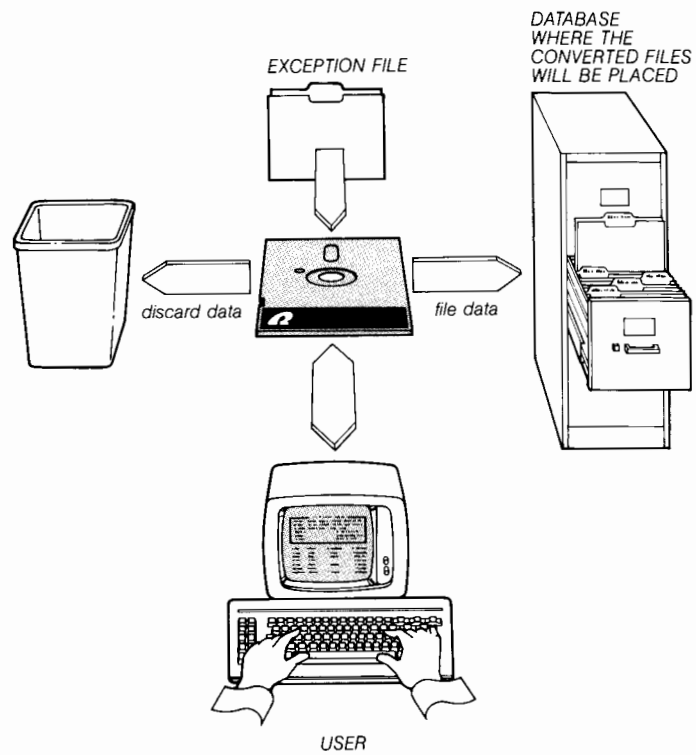


Figure 13-3 Correcting Exceptions

## Definition of Terms

The FileGateway uses the terms *file*, *field*, and *record* to identify the elements of an incoming file. R:base 5000 uses the terms *table*, *column*, and *row* after converting the files to R:base format.

Where possible, product-specific terminology is used in this chapter. In general discussions, the terms *file*, *field*, and *record* are used. Table 13-1 summarizes the terms used by various systems.

Table 13-1 Database Elements Cross-Reference

Product	Database Elements				
FileGateway	database	file	record	field	
R:base 5000	database	table	row	column	
R:base 4000	database	relation	row	attribute	
ASCII Fixed	*	file	record	field	
ASCII Delimited	*	file	record	field	
Lotus 1-2-3	*	worksheet	row/column	cell	
dBASE II	*	file/database	record	field	
PFS:FILE	1 file only	file	form	item	
VisiCalc	*	file	row/column	cell	
Multiplan	*	file	row/column	cell	

\*No comparable term.

The FileGateway can convert files from a variety of programs, each with its own method of identifying the files. A common method is to use a file extension consisting of a period and one to three characters after the file name (for example, *filename.ext*). FileGateway expects the files from other sources to have the proper file extension. Table 13-2 lists the proper extensions for each of the file types.

Table 13-2 File Type Identifier Extensions

Product	Extension	Example
R:base ASCII*	any	FILENAME.XXX
dBASE II	.DBF	FILENAME.DBF
Lotus 1-2-3	.WKS	FILENAME.WKS
PFS:FILE	none	FILENAME
VisiCalc (DIF files)	.DIF	FILENAME.DIF
Multiplan (SYLK files)	.SYL	FILENAME.SYL
ASCII	optional	FILENAME.DAT

\*R:base files are unloaded files in ASCII delimited format.



## Data Types and Formats

During the conversion process, the FileGateway automatically suggests the data type of each field and you confirm that suggestion. The data type tells the program what it may do with a piece of information. Text, for example, cannot be added or subtracted, but it can be arranged alphabetically. When the FileGateway is told that a field is TEXT type data, it knows the rules for manipulating it.

The FileGateway uses the same data types as R:base: DATE, DOLLAR, INTEGER, REAL, TEXT, and TIME. The only difference is in the DATE data type which is more flexible in R:base. Table 13-3 shows the DATE formats available in the FileGateway.

Table 13-3 FileGateway Date Formats

Date Format	Meaning	Example
YY/MM/DD	year/month/day	85/03/15
MM/DD/YY	month/day/year	03/15/85 (American)
DD/MM/YY	day/month/year	15/03/85 (European)
DD month (YY)YY	combination 1	15 March 85 or 15 March 1985
month DD, (YY)YY	combination 2	March 15, 1985 or March 15, 85
#####	Lotus "serial number" date format*	30947

\*Lotus "serial number" dates are determined by counting the number of days since January 1, 1900. The maximum internal date allowed is December 31, 1999, which is 36525. Use for Lotus conversion only.

Table 13-4 compares R:base data types to those used in other products. Note that PFS and ASCII files do not have declared data types.

Table 13-4 Database Data Types Comparison

R:base	dBASE II	Lotus	Multiplan (SYLK)	VisiCalc (DIF)
DATE	Character	Label	Text	String
DOLLAR	Number	Number	Number	Number
INTEGER	Number	Number	Number	Number
REAL	Number	Number	Number	Number
TEXT	Character	Label	Text	String
TEXT	Logical*	—	—	—
TEXT	—	Formula	—	—
TIME	Character	Label	Text	String

\*The dBASE II logical is a one-character text field.

The specific conversion procedures in this chapter include tables listing:

- Formats you use with your product
- R:base data type you would probably want for a converted column

Use these tables to see if the FileGateway guessed correctly during the conversion and, if not, to guide you in selecting the proper data type for the field being converted.

## Orientation of Spreadsheets

Before converting your spreadsheet files, consider the row and column orientation of data. You can choose to make the rows (left-to-right entries) the converted database rows or to make the columns (top-to-bottom entries) the database rows.

### SPREADSHEET ROWS AS DATABASE ROWS

If you convert a spreadsheet so that the rows are the database rows, the columns become the database columns. For example, the spreadsheet in figure 13-4 is used as a check register.

	A	B	C	D
1	Check No.	Date	Payee	Amount
2				
3	1001	12/01/84	Mutual Mortgage	\$950.00
4	1002	12/01/84	City Electric Co.	\$325.50
5	1003	12/01/84	Municipal Natural Gas Co.	\$230.31
6	1004	12/02/84	Midtown Office Supply	\$95.00
7	1005	12/02/84	Central Computer Store	\$3,560.75
8	1006	12/03/84	Federal Bank	\$4,900.25

Figure 13-4 Sample Spreadsheet Layout 1

Each of the six rows—3 through 8—contains the data for one check. If a row is to be a row in the database table, then columns A through D become the columns in the converted table. For example, the *Payee* field (column C) holds the names of all payees in the file. After the file is converted, if you ask R:base to select all items in that column, it displays a list similar to column C in your spreadsheet.

### SPREADSHEET COLUMNS AS DATABASE ROWS

You can also use the other orientation: have the column become the database row. Consider the example in figure 13-5.

	A	B	C	D	E
1					
2		JAN	FEB	MARCH	APRIL
3	Eastern Sales	\$1,200	\$2,150	\$2,300	\$1,750
4	Western Sales	\$1,110	\$2,050	\$1,900	\$2,050
5	Total Sales	\$2,310	\$4,200	\$4,200	\$3,800
6	Costs of Goods	\$1,200	\$2,100	\$2,000	\$2,000
7	Gross Margin	\$1,110	\$2,100	\$2,200	\$1,800

Figure 13-5 Sample Spreadsheet Layout 2

If you convert the spreadsheet by column, then each column of data becomes a row in the table as in figure 13-6.

MONTH	EASTSALE	WESTSALE	TOTSALES	COSTGOOD	GROSS
JAN	\$1,200	\$1,110	\$2,310	\$1,200	\$1,110
FEB	\$2,150	\$2,050	\$4,200	\$2,100	\$2,100
MARCH	\$2,300	\$1,900	\$4,200	\$2,000	\$2,200
APRIL	\$1,750	\$2,050	\$3,800	\$2,000	\$1,800

Figure 13-6 Converted R:base Table

### TOTAL AND CALCULATION FIELDS

Values such as totals and subtotals which have been calculated from other data in the worksheet do not have to be converted. R:base calculates totals and subtotals for you either directly with commands such as `COMPUTE` or indirectly through variable expressions.

If you do convert rows or columns of calculated values, it is best to orient the original file so that the row or column with totals has its own column name as shown in figure 13-6. If the file is oriented as in figure 13-5, you may get inflated answers from adding component values to the totals. For example, gross margin may be added to the total sales if you requested a total on a month column.

### SPECIFYING SPREADSHEET BOUNDARIES

A point where a row and column intersect is called a *cell* in Multiplan and Lotus 1-2-3. When you convert this worksheet with the FileGateway, you indicate the beginning and the end of the section you are converting by specifying the upper-left cell and the lower-right cell. For example, in figure 13-5, the boundaries are A2 as the upper-left corner and E7 as the lower-right corner.

### Sample Conversion File

To help you learn the FileGateway, table 13-5 shows you a sample of each data type the FileGateway can convert. All sample files contain information from a fictional company, ACE Rental, which rents lawn and garden equipment. The ACE file includes the name of each piece of equipment, corresponding item numbers, hourly and daily rental rates, time, date and length of time the equipment is rented, tax rate, and total charges.

The data in the file is divided into fields with different data types which define how the FileGateway treats the data. When you convert a file, the FileGateway chooses a data type for each field. If the FileGateway chooses incorrectly, you can change the data type with the conversion editor. The ACE Rental Co. database contains data that can be classified into all the data types R:base recognizes.

Table 13-5 Sample File Data Types

Item	Data Type	Field Description
Item Number	TEXT	Hyphenated number used for inventory control, for example 07-288-175. The number is used as a name and cannot be used for arithmetic calculation.
Tool Name	TEXT	Description of the tool, such as <i>lawn mower</i> .
Date Out Date In	DATE	Indicates DATE format. Dates can also be treated as a TEXT data type or INTEGER for Lotus "serial number" dates. The ACE file format is MM/DD/YY.
Time Out Time In	TIME	Time an item is rented out and when it is returned. The format is HH:MM:SS.
Days Out Hours Out	INTEGER	Total days and hours the equipment was rented out. INTEGER was chosen because the items are rented by the whole day plus additional whole hours. All arithmetic operations can be performed.
Day Rate Hour Rate	DOLLAR	Daily and hourly rental rates for each item. The DOLLAR data type tells the program that two decimal places follow a whole number. Answers are displayed with the decimal places lined up.
Tax Rate	REAL	Computes the amount of tax a customer owes. As a REAL data type the number can have decimals and all arithmetic operations can be performed.
Total Due	DOLLAR	Total charges based on the total number of days and hours out multiplied by the daily or hourly rate plus the computed tax.

Table 13-6 lists the sample files provided on the FileGateway disk and other information you need for converting the sample file to R:base format.

Table 13-6 Sample File Names

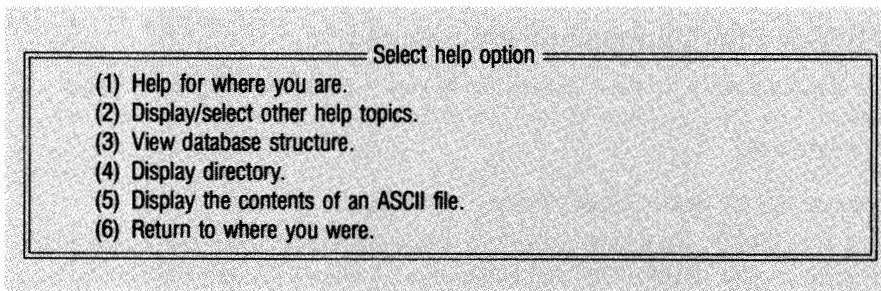
Name of File	Program or File Type	Other Information Required
VISICALC.DIF	DIF file (from VisiCalc)	none
MULTIPLA.SYL	SYLK file (from Multiplan)	coordinates R9C1 and R13C12
ASCII.FIX	ASCII file with fixed fields	none
ASCII.DEL	ASCII file with delimiters	delimiter is comma ( , )
LOTUS.WKS	Lotus 1-2-3 file	Label row is 5 coordinates B7..M11
DBASEII.DBF	dBASE II file	none
PFSFILE	PFS:FILE	none

## GENERAL CONVERSION PROCEDURES

### Help Menu

The FileGateway offers you on-line help whenever you ask. When you do not understand how to respond to the FileGateway prompts, ask for help by pressing [F10] (function key 10) at any menu.

The help options menu is displayed as shown below:



To select an option from a FileGateway menu, use the cursor keys to highlight the option you want. Then press [ENTER] to select the option or press the number key corresponding to the option you want. The options are described in table 13-7.

Table 13-7 Help Menu Options

Option	Explanation
1	Displays a help screen associated with the task at hand.
2	Displays an additional help menu containing information on topics of interest such as program overview, input file types, vocabulary translation, and duplicate file names.
3	Displays the database structure.
4	Displays a DOS directory (file names, wildcards and paths optional). Use this option to locate the original file name, if needed.  The directory feature on the help menu is slightly different than standard DOS. When you have a multilevel directory, you must specify the entire path rather than just the subdirectory level you want. For example, if you have a path like this, <code>C:\sub1\sub2\sub3</code> , and your current subdirectory is <code>sub2</code> , you must specify <code>C:\sub1\sub2\sub3\</code> to display a directory of <code>sub3</code> not just <code>C:\sub3\</code> . Also, notice that you must start and end the subdirectory list with the backslash character (\)—for example <code>C:\sub1\</code> not <code>C:\sub1</code> .
5	Displays the contents of an ASCII file. This option shows the contents of the file as it is stored.
6	Returns to where you were.

## General File Conversion Procedures

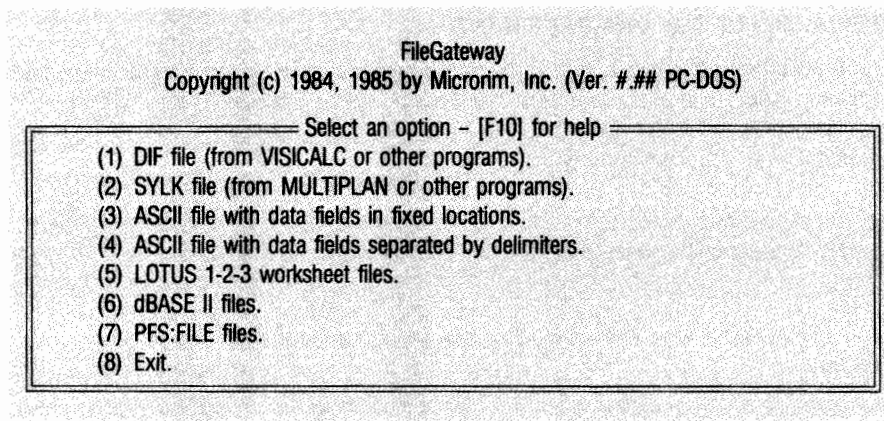
The section gives a detailed explanation of each step taken in the conversion process. See “Specific Conversion Procedures” for the specific product file you are converting.

### STEP 1: START THE FILEGATEWAY

Type GATEWAY at the DOS prompt or select GATEWAY from the R:base 5000 main menu. See chapter 1 for full information. The two FileGateway files, GATEWAY.EXE and GATEWAY.DAT, must be on the default drive.

**STEP 2: SELECT FILE TYPE**

The FileGateway displays the file type selection menu:



Select the original file type by using the cursor keys to highlight the option you want then press [ENTER] to select.

**STEP 3: ENTER THE ORIGINAL FILE NAME**

If the selected file type is Lotus, dBASE II, or SYLK format, the FileGateway displays a list of files on the default directory with the proper file extension, *Other* to allow you to enter a file name from a different drive or directory, and *Exit* to return you to the file type selection menu. Select a file name, *Other*, or *Exit* by using the cursor keys to highlight the selection then pressing [ENTER].

The FileGateway prompts for the original file name as follows if the file type is an ASCII or PFS file or you selected *Other* above:

Enter the name of the <file type>  
file to be converted or [F10] for help:

Enter the name of the original file, press [F10] for help, or press [ESC] to return to the file type selection menu.

Enter the name of the original file using this format: *d:\path\file.ext* where *d*: is the optional drive designation, *path* is the optional path name (DOS version 2.0 or higher is required), *file* is the original file name, and *.ext* is the standard extension for the original DOS file—for example, *b:\wrksheet\lotus.wks*.



If you are practicing with the sample files provided on the FileGateway disk, the file names are shown in table 13-6 and are provided in the “Specific File Conversion” sections.

#### **STEP 4: ENTER THE DATABASE NAME**

The FileGateway displays a list of the databases available on the default drive and directory. Select one of the displayed databases, select *Other* to enter another database name, or select *Exit* to return to the file type selection menu. If you selected *Other*, the FileGateway prompts

Enter the name of the database to be used or [F10] for help:

Enter the name of your database. This can be an existing database or a new database.

The database name is in the form *d:\path\dbname* where *d:* is the optional drive designation, *path* is the optional subdirectory path, and *dbname* is a seven character or less database name. If needed, you can press [F10] to display the help menu then select the *Display DOS Directory* option to display a directory.

Database names are limited to seven characters plus two characters for the drive designation, and a subdirectory path. For example, *b:receipt* or *d:\rbase\mydbs\transx*.

Unlike DOS file names, you cannot assign extensions to database names. They are assigned the extension .RBS automatically. Whenever you refer to a database, either in R:base or the FileGateway, you do not include the extension.

#### **STEP 5: ENTER NAME OF THE CONVERTED TABLE**

The FileGateway displays a list of the available tables. Select one of the tables, (*NEW*) to enter a new table name, or press [ESC] to return to the file type selection menu. If you selected (*NEW*) or no tables were available in the database, the FileGateway prompts

New name for the table within the database:

The table name must be eight characters or less and must not have an extension. The FileGateway displays the original file name. Press the [ENTER] key to accept this as the converted table name or enter the name for the converted table. If this is a new table name, continue to step 7.

If you receive a Not enough disk space message, see step 6.

### Choosing an Existing Table

If you choose a table name that already exists in the database, the existing table name menu appears with five options:

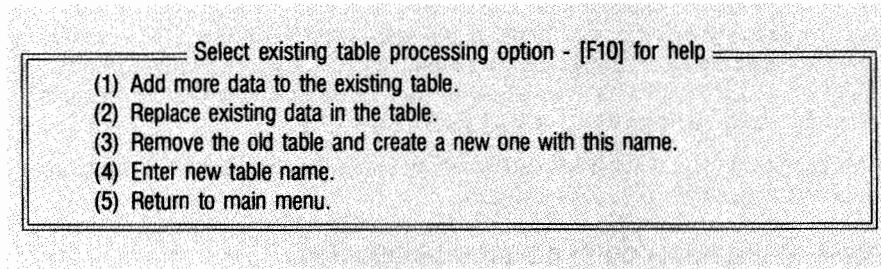


Table 13-8 Existing Table Name Menu Options

Option	Explanation
1	Adds more data to the existing table. The information in the file you convert is added to the end of the existing table. The information that already exists is not changed, but the table is enlarged.
2	Replaces the existing data in the table. Your new information is recorded over the old information. If you select this option, the structure of the old table is maintained and the new data must fit into that structure.
3	Removes the old table and creates a new one with this name. The old table and structure are completely erased, and the file you are converting is treated like any new table. Continue to step 7.
4	Allows you to change the name of the table so that it does not conflict with the name of an existing table. You are prompted to type in a new name that does not conflict with the existing names. Continue to step 7.
5	Return to the file type selection menu.

If you select option 1 or 2, the structure of the new data must exactly match the structure of the old data (the order, data type, and length of the columns). Before the data is converted, the FileGateway prompts you to enter the date format, coordinates for spreadsheet type files, start and end markers for ASCII fixed field files, or the delimiter character for ASCII delimited files. You do not edit the input file characteristics as you would if you were creating a new file (as shown in steps 7 through 9). Continue to step 9 for DIF, SYLK, ASCII Delimited, Lotus, dBASE II, and PFS:FILE conversions. See the "Specific Conversion Procedures" section for more information on ASCII Fixed Field files.

Select the option you want from this menu by using the cursor keys to highlight the option then press [ENTER] to select.

#### **STEP 6: MAKE ROOM ON THE DISK IF NECESSARY**

If you try to convert a file that is too large for the currently available disk space, you receive the warning message:

Not enough disk space for converted file on the database disk.

The FileGateway has checked the size of the file to be converted, estimated the size of the resulting database, and found the available disk space to be insufficient. You are not allowed to convert the file. The file type selection menu is redisplayed.

Check the length of the file and the available disk space. To remedy this problem, you have two options:

- Delete files to make room for the converted file
- Put the converted file on another disk

After correcting the problem, you must restart the FileGateway and begin at step 1.

#### **STEP 7: SET SPECIFIC PARAMETERS IF NECESSARY**

Follow detailed instructions in the next section "Specific Conversion Procedures" for the file type you want to convert.

**STEP 8: EDIT FIELD CHARACTERISTICS**

The FileGateway displays the file conversion editor screen with the first record of the original file as shown below for the sample file:

File Conversion Editor	
The listing below shows the fields found in your input file along with the most likely type for that field, and a sample of the data found.	
To make changes in the name, type, and/or length of a field, move the cursor to that field and type in the new information.	
Up, Dn, Lt, Rt, PgUp, PgDn, Home, End	Moves the cursor between fields
+	Gets next record of sample data
F10	Help
ESC	When changes are complete

Field Name	Type of Data	Maxlength	Sample Values
FLD08001	TEXT	10	05-726-692
FLD08002	TEXT	20	Skill Saw
FLD08003	DATE		07/15/84
FLD08004	TIME		09:15:00
FLD08005	DATE		07/15/84
FLD08006	TIME		15:40:00
FLD08007	TEXT		XYZ
FLD08008	INTEGER		7
FLD08009	DOLLAR		15.00

Edit the field names, data types, and maximum lengths.

The values in the record are displayed in the column at the right side of the screen. The FileGateway has guessed the data type and maximum length of each field. You may edit the field characteristics to define the input values. Once converted, the tables then use the default R:base formats. Make sure that the FileGateway understands the nature of each field being converted by properly defining the type of data and the maximum length of TEXT fields.

- Arrow keys move the cursor to the row/column you want to edit. The [ENTER] key also moves the cursor to the next characteristic to be edited.
- [Home] moves the cursor to the top of the record.
- [End] moves the cursor to the bottom of the record.
- [+] displays the next record of sample data. This helps you validate the data type and maximum length.
- [PgDn] displays any undisplayed fields (nine data fields are displayed at a time). The [PgUp] key scrolls the fields back up.

You need only edit the characteristics once. The FileGateway assumes that the field name, data type, and maximum length define the characteristics of that field for all records in the original file.

You may change the field name and type of data by typing over the displayed type that the FileGateway has guessed. To change a data type, you need only enter the first one or two unique letters of the data type. If the type of data is TEXT, you can also change the maximum length. Sample data values cannot be changed.

Field names are best changed to a descriptive column name. The name can be any text combination of up to eight characters. Field names longer than eight characters are truncated except PFS:FILE field names which must be reentered.

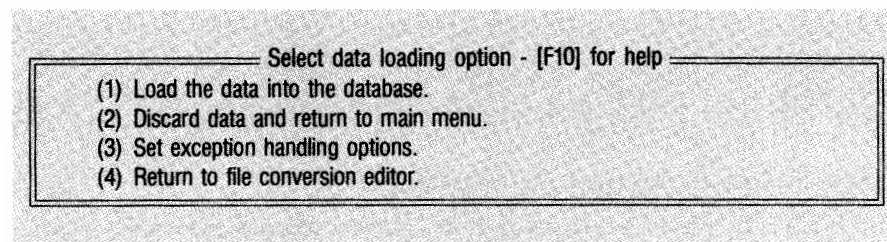
If your database contains another table with a column name the same as any entered for this file, the data type and maximum length (if text) must match. If the data type in the duplicate column does not match, you must set your data types to match or the FileGateway prompts you to change the newly entered column name.

Note that the file conversion editor is not a true data editor. You are changing the field characteristics, not editing them in the usual data processing sense.

Press [ESC] when you are finished editing the field characteristics.

#### **STEP 9: LOAD THE DATA**

The FileGateway now displays the database load menu:



```

      _____ Select data loading option - [F10] for help _____
      |
      | (1) Load the data into the database.
      | (2) Discard data and return to main menu.
      | (3) Set exception handling options.
      | (4) Return to file conversion editor.
      |
      |_____

```

Select the option you want by using the cursor keys to highlight the option, then press [ENTER]. Table 13-9 describes the database load options.

Table 13-9 Database Load Menu Options

Option	Explanation
1	Loads the data. Exceptions are diverted to the temporary exception file (EXCEPT.DAT is the default exception file name).
2	Stops the process and redisplay the file type selection menu. No data is loaded; no files are created.
3	Goes to the exception handling options menu which allows you to 1) change the default exception file name or 2) turn off exception handling. See "Exception Handling Options" for details.
4	Returns to the file conversion editor for any additional changes or review. Go back to step 8.

If you select option 1 and your file contains a date field, the system displays the date format selection menu:

Select date format - [F10] for help

- (1) MM/DD/YY.
- (2) YY/MM/DD.
- (3) DD/MM/YY.
- (4) DD month (YY)YY e.g. 15 March 85 or 15 March 1985.
- (5) month DD, YYYY e.g. March 15, 1985.
- (6) Lotus 'serial number' (number of days since Jan 1, 1900).
- (7) Return to file conversion editor.



Select the option you want by using the cursor keys to highlight the option, then press [ENTER] to select. Select the date format that corresponds to the format in the original file. If you are not sure, look at the field values in several records. If the date format is not stored in a valid format, select a valid format from the date select menu and the FileGateway will attempt to convert the invalid format to the valid format. If not successful, then you have to change the date data types to TEXT. To do this, select option 7.

The FileGateway then loads the data and displays the load information:

```
Begin adding data to the database
Rows loaded:
Database:      4          Exception file:      1
Finished adding data to the database
      1 FIELDS INVALID OUT OF TOTAL OF 5 RECORDS
Press any key to continue . . . . .
```

Press any key to continue. If any records were sent to the exception file continue to step 10. Otherwise, you are done.

The FileGateway keeps a tally of the number of records loaded into the database and the number of records sent to the exception file because they could not be converted.

If no exceptions were stored in the exception file, then the conversion process is finished and the file type selection menu is redisplayed. Either select another file type for conversion or exit the program (see “Leaving the Program” in this section).

#### **STEP 10: CORRECT AND LOAD EXCEPTIONS**

If any records cannot be loaded because of mismatches in data type or length, they are considered *exceptions*. These exceptions are loaded into the exception file (EXCEPT.DAT or your own exception file name if you choose to change it).

If there are any exception records in the data, the FileGateway displays:

Field Name	Type of Data	Maxlength	Sample values
item-num	TEXT	10	05-726-692
toolname	TEXT	20	Skill Saw
dateout	DATE		07/15/84
timeout	TIME		09:15:00
datein	DATE		07/15/84
timein	TIME		15:40:00
daysout	INTEGER		XYZ
hoursout	INTEGER		7
dayrate	DOLLAR		15.00

You can deal with exception records either individually or all together as described below. If you choose to correct individual records, you have the following options:

- Correct the exception data and load the record
- Load the displayed record with the exception changed to a null value
- Delete the record with the exception data

At any time while correcting, loading, or deleting individual records, you can choose to switch methods and deal with the rest of exception records together.

If you choose to deal with all of the exception records, press [ESC] at any record to display the exception loading options. You have the following options:

- Load all records with exception data changed to a null value
- Discard all records with exception data



**Correcting, Loading, or Deleting Individual Records**

**Correcting a Record.** If you want to correct the data, move the cursor to the highlighted exception fields and change the field values. Press [ENTER] at the end of each field edit.

Data with exceptions is displayed one record at a time. Fields which could not be loaded are highlighted. An exception might occur if, for example, you designated a field as an INTEGER data type and one of the records had text as data. In the example, the DAYSOUT field is INTEGER but the data is XYZ, therefore, the FileGateway has flagged the record as an exception. You would edit the value of the DAYSOUT field to an integer, or you could allow the FileGateway to assign a null value to the DAYSOUT field.

Nine fields are displayed on the screen at one time. To see more fields, press the [PgDn] or [↓] key.

When you finish editing a field press [ENTER], then use the cursor keys to move to the next exception field, if any.

When you are done editing the highlighted exception fields, move the cursor to an unhighlighted field and press [+] to load the record. The next record is displayed until all exception records are corrected or you press [ESC].

**Loading a Record.** Pressing [+] loads the record into your database and converts any uncorrected exceptions to null values. A null value means *no entry*. The record is placed at the end of the table, but no value is entered in the column with the exception. The null value is represented by the symbol -0-.

Records are added to the bottom of the database, therefore, the records are not in the same order as they were in the original file.

Press [ESC] at any time to return to the exception loading menu.

**Deleting a Record.** Remove any individual record with exceptions by pressing [Del].

**Loading or Discarding All Records**

If you choose to deal with all exception records together and press [ESC] at any exception record, the exception loading menu is displayed:

<p>Select exception loading option - [F10] for help</p> <ul style="list-style-type: none"><li>(1) Change exception field in remaining records to nulls and load.</li><li>(2) Do not load—Discard the rest of the exception file.</li><li>(3) Return to exception handling editor.</li></ul>
---

Select the option you want by using the cursor keys to highlight the option, then press [ENTER]. Table 13-10 describes the exception loading options. Both options 1 and 2 return to the file type selection menu after use.

Table 13-10 Exception Loading Menu Options

Option	Explanation
1	Converts all the exception records and adds them to the database table. Any fields which cannot be converted or were not edited are set to a null value (null means no entry and is represented by the null symbol -0-).
2	Deletes the exception file without loading any unedited records. Records which were edited and loaded by typing [ + ] are already written to the database table.
3	Returns to the exception handling editor. Go back to step 8 for editing procedures.

## Exception Handling Options

If you choose to set exception handling options (option 3 from the database load menu), the FileGateway displays the exception handling menu as shown below.

The exception file is a temporary file which is used to store records that can not be converted and loaded with the rest of the original file. After the records in it are corrected and loaded, or discarded, it is automatically erased.

Select exception handling option - [F10] for help

- (1) Change exception file (default is EXCEPT.DAT).
- (2) Turn off exception handling.
- (3) Return to previous menu.

Table 13-11 describes the exception handling options.

Select the option you want by using the cursor keys to highlight the option, then press [ENTER]. After returning to the database load menu, select option 1. Your table is loaded and the temporary exception file is deleted.

Table 13-11 Exception Handling Menu Options

Option	Explanation
1	<p>Changes the exception file name or its drive location. The default name given to the exception file is EXCEPT.DAT on the default drive. You can change the name and drive of the file. You only need to do this if you already have an EXCEPT.DAT file on your disk or you want to use a different drive.</p> <p>The FileGateway prompts for the new exception file name. Type in the new name with the drive and directory specification (for example, <i>b:\mypath\filerr</i>s) and press [ENTER].</p>
2	<p>Turns off the exception handling function for this file only. Records containing exceptions are not written to any file or saved in any way.</p> <p>Options 1 and 2 both redisplay the database load menu after use.</p>
3	<p>Returns you to the database load menu without changing the exception handling options.</p>

## Leaving the Program

Before leaving the program you can check the structure of the new tables by pressing [F10] for help and then select option 3 to display the database structure.

To leave the FileGateway, select option 8 from the file type selection menu. You are returned to the R:base 5000 main menu or to the operating system. From the R:base 5000 main menu you can start R:base, or other programs.

If you created a practice database and want to delete it, use the DOS or R:base *erase* command as follows:

```
ERASE d:\path\dbname?.RBS
```

where *d:* is the drive where you stored the practice database, *path* is the optional path, *dbname* is the name you gave the practice database, *?* is the wildcard character used to allow you to erase all three DOS files created for the database, and *.rbs* is the required extension assigned by the FileGateway. The same method can be used to delete any database you create whether through the FileGateway or R:base.

## SPECIFIC CONVERSION PROCEDURES

### VisiCalc Worksheets in DIF File Format

#### WHAT YOU NEED TO KNOW BEFORE CONVERSION

When you store a worksheet in DIF format, be sure to carefully consider how you want the final database table to look. If you want it to match the orientation of the original worksheet, save it to a DIF file oriented by column. If you want the worksheet reversed then save it by row.

If you save the worksheet by column, each row (left-to-right) becomes a record in the DIF file. For example, in figure 13-7, the column names are the names of the months. Each DIF record contains data for all four months. For example, the first DIF record contains the following:

Eastern Sales	\$1,200	\$2,150	\$2,300	\$1,750
---------------	---------	---------	---------	---------

If you save the worksheet in the DIF file by row, each DIF record contains the month, eastern sales, western sales, total sales, cost of goods, and gross margin. For example, the DIF record from column B contains the following:

JAN	\$1,200	\$1,110	\$2,310	\$1,200	\$1,110
-----	---------	---------	---------	---------	---------

	A	B	C	D	E
1					
2		JAN	FEB	MARCH	APRIL
3	Eastern Sales	\$1,200	\$2,150	\$2,300	\$1,750
4	Western Sales	\$1,110	\$2,050	\$1,900	\$2,050
5	Total Sales	\$2,310	\$4,200	\$4,200	\$3,800
6	Costs of Goods	\$1,200	\$2,100	\$2,000	\$2,000
7	Gross Margin	\$1,110	\$2,100	\$2,200	\$1,800

Figure 13-7 VisiCalc Worksheet Example

**CONVERTING A VISICALC WORKSHEET TO DIF FORMAT**

To use R:base with a VisiCalc worksheet, you need to store your worksheet in a DIF file. To do this, use the Save DIF option of the VisiCalc storage command (/S#S). If you have another product that uses the DIF format, see the other product's user manual for procedures.

If the worksheet is divided into areas of unrelated data, you should save each area in a separate DIF file. When you use the VisiCalc STORAGE command, specify the coordinates for the area of the worksheet to convert.

The VisiCalc program saves a worksheet either by row or by column. Your choice of which method to use depends on how your worksheet is arranged.

The following procedure describes how to convert a VisiCalc worksheet into DIF file format.

1. Load VisiCalc and display the worksheet you want to save.
2. Place the cursor at the upper-left boundary of the worksheet area you want to save. Do not include labels.
3. Type the following sequence of commands (VisiCalc prompts are displayed). Do not press [ENTER] between commands.

```
/
S
#
S
d:filename
[ENTER]
```

where *d:* is the drive and *filename* is the DIF file name.

4. Place the cursor at the lower-right boundary of the worksheet area you want to save or enter the coordinates. Press [ENTER].
5. Visicalc prompts for row or column. Type *R* or press [ENTER] to store by row. Type *C* to store by column.

The DIF file should have data only. Do not include column or row headings when translating the VisiCalc file into DIF format. You add labels during the FileGateway conversion process. See the *VisiCalc User's Guide* for information on the DIF file format.

## CONVERTING A DIF FILE TO R:BASE FORMAT

Following is a short form of the procedure you must use to convert DIF files to an R:base format. For details, see the corresponding step in the "General File Conversion Procedures" section.

1. Start the FileGateway.
2. Select 1 at the file type selection menu.
3. Enter the original file name with extension. Include the drive and subdirectory, if needed. The practice file is named *visicalc.dif*.
4. Enter the database name. Include the drive, if needed.
5. Enter the converted table name or press [ENTER] to use the original file name.  
If this is a duplicate table name, either rename the new table, add this data to the existing table, replace existing data, or delete the old table.
6. Make room on the disk if the file is too large for the specified drive. To do this you must exit from the FileGateway.
7. DIF files have no special parameters. Go to step 8.
8. Edit the field names, data types, and maximum lengths. Use table 13-12 to properly enter data types in the FileGateway. Press [ESC] when you are finished editing the field characteristics.

Table 13-12 VisiCalc Data Formats

Format	Example	R:base Type
General	1000.234	REAL
Integer*	101	INTEGER*
Dollars-and-cents	100.25	DOLLAR
Graph	212†	TEXT†

\*The VisiCalc integer format rounds off a number. For example, 101.1 is shown as 101. The R:base INTEGER data type does not round. The number entered for this type must already be a whole number.

†The graph format adds asterisks to integers to show the digits that were rounded off. R:base INTEGER and REAL data types allow only numerical characters to be entered. To keep the graph display, choose the TEXT data type.

9. Load the data into the database. Select a date format, if needed. If there are no exception records, you are done.

10. Correct and load exceptions.

Edit the highlighted exception cells.

Press [ + ] to load the edited record. The next record, if any, is displayed. If there are no more exception records, the file type selection menu is redisplayed. You are done.

Press [ + ] without editing the exception data. The record is loaded with a null value in the exception field. The next record, if any, is displayed. If there are no more exception records, the file type selection menu is redisplayed. You are done.

Press [D] to delete the displayed exception record. It is not loaded. The next record, if any, is displayed.

Press [ESC] to display the exception loading menu. Options 1 and 2 return to the file type selection menu. Select another file type to process or select option 8 to quit.

## **Multiplan SYLK Files**

### **WHAT YOU NEED TO KNOW BEFORE CONVERSION**

Multiplan spreadsheets must first be converted to the SYLK file format before you use the FileGateway. The SYLK file format is row oriented and must be read by the FileGateway one row at a time. Thus, each row becomes a single database row.

Spreadsheets fall into two general orientation categories: row and column. The examples below illustrate these two kinds of spreadsheets. A discussion of how the FileGateway treats them follows the examples.

#### **Row Orientation**

When the FileGateway converts a spreadsheet like the one in figure 13-8, each row—with information for each check—becomes a separate database row. The field headings are the descriptive titles in row 1.

	1	2	3	4
1	Check No.	Date	Payee	Amount
2	1001	12/01/84	Mutual Mortgage	\$950.00
3	1002	12/01/84	City Electric Co.	\$325.50
4	1003	12/01/84	Municipal Natural Gas Co.	\$230.31
5	1004	12/02/84	Midtown Office Supply	\$95.00
6	1005	12/02/84	Central Computer Store	\$3,560.75

Figure 13-8 Multiplan Spreadsheet Example 1

In response to the FileGateway's prompt, Enter the coordinates of the upper-left corner of your data area, you enter *R2C1* for row 2, column 1 for this example.

Note that the field labels in row 1 are ignored at this time. These labels are entered later when you use the file conversion editor. Also note that if column 1 contains labels such as *Bank Name*, these labels are considered to be data and thus are included in the data area.

### Column Orientation

The column-oriented spreadsheet in figure 13-9 uses each column for a single row of the information for the month.

	1	2	3	4...	13	14
1		JAN	FEB	MAR	DEC	TOTAL
2	SALES	\$2,310	\$4,200	\$4,500	\$5,600	\$49,610
3	COST OF SALES	\$1,110	\$2,100	\$2,300	\$3,000	\$28,400
4	GROSS MARGIN	\$1,200	\$2,100	\$2,200	\$2,600	\$21,210

Figure 13-9 Multiplan Spreadsheet Example 2

Because the SYLK file format saves data in rows, the FileGateway cannot convert this spreadsheet to a column-oriented database. Therefore, it is recommended that you not undertake the conversion of column-oriented Multiplan spreadsheets.



**File and Record Length**

The FileGateway can process records up to 1600 characters long. If the Multiplan spreadsheet contains rows longer than 1600 characters, the FileGateway may not be able to convert the SYLK file. If possible, you should divide the spreadsheet vertically before transferring it to the SYLK file format. If you do this, make sure you add a common field to the second and subsequent files. For example, in the first example (figure 13-8), the check number could be duplicated in the additional files to form the common linking field between the files. Before dividing the spreadsheet, be sure to read "Linking Fields" below.

**Linking Fields**

When a spreadsheet is divided into separate tables, a link (common characteristic) can be set up in each of the tables so that relationships are formed. In the row-oriented example, the link is column 1-the check numbers. Before transferring the Multiplan file to SYLK format, divide the spreadsheet and add the linking field to each separate spreadsheet file.

**File Conversion Editor**

If you have a month field (such as JAN, FEB, MAR), you should change the data type from TEXT to DATE and select a suitable date format such as MM/DD/YY.

**Exception Handling Editor**

The month fields changed to DATE type in the file conversion editor are displayed as exception fields. Change the dates to real date formats. For example, change JAN to 01/01/84. Use either the first or last day of the month. Alternatively, you can change invalid date formats in Multiplan before you convert the file to SYLK format.

**CONVERTING A MULTIPLAN WORKSHEET TO SYLK FORMAT**

To use R:base with a Multiplan worksheet, open the file with Multiplan and note the coordinates that define the data area you want to convert. SYLK files can be read in row orientation only. Store the worksheet in a SYLK file. During the FileGateway conversion process you are prompted for the coordinates of the upper-left and lower-right corners of the data area. See "Orientation of Spreadsheets" in this chapter for a discussion of data coordinates.

SYLK files are created within Multiplan by selecting the *Symbolic* option of the TRANSFER command. Use the following procedure to convert Multiplan files to SYLK format.

1. Load Multiplan and load the worksheet you want to transfer.
2. Press [T] for the TRANSFER command.
3. Press [O] for Options.
4. Press [S] for Symbolic (SYLK). Then press [ENTER].
5. Press [T] for the TRANSFER command.
6. Press [S] to save the file in SYLK format.
7. Enter *d:filename.syl* where *d:* is the drive, *filename* is the file you want to transfer to SYLK format, and *.syl* is the required extension. Be sure that the file has an extension of .SYL before attempting to convert it with the FileGateway. If not, rename the file with the RENAME command.

### CONVERTING A SYLK FILE TO R:BASE FORMAT

Following is a short form of the procedure you must use to convert SYLK files to an R:base format. For details, see the corresponding step in the "General File Conversion Procedures" section.

1. Start the FileGateway.
2. Select 2 at the file type selection menu.
3. Enter the original file name with extension. Include the drive and subdirectory, if needed. The sample file is named *multipla.syl*.
4. Enter the database name. Include the drive, if needed.
5. Enter the converted table name or press [ENTER] to use the original file name.  
If this is a duplicate table name, either rename the new table, add this data to the existing table, replace existing data, or delete the old table.
6. Make room on the disk if the file is too large for the specified drive. To do this you must exit from the FileGateway.
7. The FileGateway prompts for the SYLK file coordinates:

Enter the coordinates of the upper-left corner of your data area  
or [F10] for help. (Use Multiplan notation, i.e. RnCn)

Enter the upper-left corner coordinate in the Multiplan standard notation Row#Column#. In the sample file, the upper-left coordinate is R9C1.

The coordinates define the data area of the table only. Do not include field labels. You enter column names when you edit the file for field characteristics.

The FileGateway then prompts:

Enter the coordinates of the lower-right corner of your data area  
or [F10] for help. (Use Multiplan notation. i.e. RnCn)

Enter the lower-right corner coordinate in the Multiplan standard notation Row#Column#. In the sample file, the lower-right coordinate is R13C12.

8. Edit the field names, data types, and maximum lengths. Use table 13-13 to properly enter data types in the FileGateway. Press [ESC] when you are finished editing the field characteristics.

Table 13-13 Multiplan Data Formats

Format	Example	R:base Type
Continuous	(Long text)	TEXT
Scientific	2.1E6	REAL
Fixed point	20.123	REAL
General	2000.912	REAL
Integer*	2001	INTEGER*
Dollar	\$100.25	DOLLAR
Bar graph	***	TEXT
Percent	15%	REAL

\*The Multiplan integer format rounds off a number. For example, 101.1 is shown as 101. The R:base INTEGER data type does not round. The number entered for this type must already be a whole number.

9. Load the data into the database. Select a date format, if needed. If there are no exception records, you are done.

10. Correct and load exceptions.

Edit the highlighted exception cells.

Press [ + ] to load the edited record. The next record, if any, is displayed. If there are no more exception records, the file type selection menu is redisplayed. You are done.

Press [ + ] without editing the exception data. The record is loaded with a null value in the exception field. The next record, if any, is displayed. If there are no more exception records, the file type selection menu is redisplayed. You are done.

Press [D] to delete the displayed exception record. It is not loaded. The next record, if any, is displayed.

Press [ESC] to display the exception loading menu. Options 1 and 2 return to the file type selection menu. Select another file type to process or select option 8 to quit.

## ASCII Files With Fixed Fields

### WHAT YOU NEED TO KNOW BEFORE CONVERSION

An ASCII file with fixed fields stores data in fixed locations as if in tabular form. For example, figure 13-10 illustrates the positions of fields in a small fixed field file.

Column —	123456789012345678901234567890
	-----
	1234682TOPEKA 89.11 20
	683 CONSTANTINOPLE. 97 21
	5 BELLEVUE 100.22110

Figure 13-10 ASCII Fixed Field File Layout

When the FileGateway converts these files, the data is displayed in a continuous line with fields starting in specific locations. For example, the city field above always starts in the eighth column for each record.

You provide marks for the FileGateway to divide the line into individual fields. You do not have to mark the entire file, only the first record. The FileGateway uses the marked record as a model for converting the entire file.

In the sample data above, you want to be sure that the field marks are placed properly between the third and fourth fields. If the line 5 BELLEVUE 100.22110 is the first record, you may not be able to tell that the value in field 3 is 100.22 and the value in field 4 is 110. If there is any doubt, look at additional records, other reference materials, or ask a resource person to make sure.

The only preparation you need for converting an ASCII file in fixed field format is to become sufficiently familiar with the file format to correctly mark the fields to complete the conversion process.

The FileGateway expects a [RETURN] to mark the end of individual ASCII records.

#### **CONVERTING AN ASCII FIXED FIELD FILE TO R:BASE FORMAT**

Following is a short form of the procedure you must use to convert ASCII fixed field files to an R:base format. For details, see the corresponding step in the "General File Conversion Procedures" section.

1. Start the FileGateway.
2. Select 3 at the file type selection menu.
3. Enter the original file name with extension. Include the drive and subdirectory, if needed. The practice file is named *ascii.fix*.
4. Enter the database name. Include the drive, if needed.
5. Enter the converted table name or press [ENTER] to use the original file name.

If this is a duplicate table name, either rename the new table, add this data to the existing table, replace existing data, or delete the old table.

If you are adding an ASCII fixed field file to an existing table, you must remark the data fields. After you mark the data fields (as described in step 7), the FileGateway displays the following menu along with the first record from the input file:

----- Select data check option - [F10] for help -----

<ul style="list-style-type: none"><li>(1) Load the data into the database.</li><li>(2) Load another record of data from the input file.</li><li>(3) Page down in the data to see the remaining fields if any.</li><li>(4) Discard the data and return to field marking display.</li><li>(5) Discard the data and return to previous menu.</li></ul>
---

If you select option 1, the data is loaded and you are done. Option 2 displays the next record from the input file. Option 3 displays additional fields from the currently displayed record. Option 4 discards the data as displayed and allows you to remark the fields. Option 5 discards the data and returns to the duplicate table name menu.

6. Make room on the disk if the file is too large for the specified drive. To do this you must exit from the FileGateway.
7. The ASCII Fixed Field Input screen appears with the first record of the file. Table 13-14 describes the keys used to mark the data.

Place the cursor at the beginning of the first field (if it is not already there). Mark the beginning of the field by pressing [S].

Move the cursor to the end of the first field. Mark the end of the field by pressing [E]. If a field is a single character, enter the start character S only.

Continue to mark the rest of the fields in the record in the same way. Be sure to mark fields clear to the end of the record. Use the [→] key to move the cursor past the initial screen display.

A column counter located above the data continuously displays the location of the cursor. This is useful for lines of data longer than 80 characters.

If you are marking the sample file, the first 80 columns of your data looks like this:

Column 1									
05-726-692Skill Saw	07/15/8409:15:00	07/15/8415:40:00	0	7	15.00				
S	ES	ES	ES	ES	ES	ES	ES	ES	E

8. Edit the field names, data types, and maximum lengths. Press [ESC] when you are finished editing the field characteristics.
9. Load the data into the database. Select a date format, if needed. If there are no exception records, you are done.
10. Correct and load exceptions.

Edit the highlighted exception fields.

Press [+] to load the edited record. The next record, if any, is displayed. If there are no more exception records, the file type selection menu is redisplayed. You are done.

Press [+ ] without editing the exception data. The record is loaded with a null value in the exception field. The next record, if any, is displayed. If there are no more exception records, the file type selection menu is redisplayed. You are done.

Press [D] to delete the displayed exception record. It is not loaded. The next record, if any, is displayed.

Press [ESC] to display the exception loading menu. Options 1 and 2 return to the file type selection menu. Select another file type to process or select option 8 to quit.

ASCII Fixed Field Input	
You must specify the field length for ASCII fixed fields. To mark the fields of the data items, place an 'S' under the first column and an 'E' under the last column of each field. Single column fields can be marked with an 'S' alone.	
Lt, Rt, Ctl-Lt, Ctl-Rt . . .	Moves cursor
+ . . . . .	Gets a new row of sample data
S . . . . .	Indicates the start of a field
E . . . . .	Indicates the end of a field
SPACE . . . . .	Deletes a start/end indicator
D . . . . .	Displays the selected field
Del . . . . .	Deletes a space or start/end indicator
ESC . . . . .	When complete

Table 13-14 Keys Used to Mark ASCII Fixed Fields

Key	Explanation
[S]	Marks the start of a field
[E]	Marks the end of a field
[→] [←]	Moves the cursor one space on the current line
[Ctrl] [→]	Moves the cursor to the end of the line
[Ctrl] [←]	Moves the cursor to the beginning of the line
[+]	Displays the next logical record
[ESC]	Ends the data marking session
[space bar]	Deletes S and E markers at the cursor location
[Del]	Deletes the space, S or E to the left of the cursor
[D]	Displays the selected field's data type, length and data value at the bottom of the screen

## ASCII Files With Delimiters

### WHAT YOU NEED TO KNOW BEFORE CONVERSION

The only preparation you need for conversion of an ASCII file in delimited format is to become sufficiently familiar with the file format to complete the conversion process.

For example, you must know the delimiter character used to mark the separation of fields. If you are not sure, use Help option 5 to display the contents of an ASCII file (see "Help Menu" in this chapter). You should also be familiar with the contents of the file so you will be able to add field names and correct data types during the conversion process.

You can also create and update ASCII delimited files with a word processor or editor, if desired.

The FileGateway expects a [RETURN] to mark the end of individual ASCII records.

### CONVERTING AN ASCII DELIMITED FILE TO R:BASE FORMAT

Following is a short form of the procedure you must use to convert ASCII delimited files to an R:base format. For details, see the corresponding step in the "General File Conversion Procedures" section.

1. Start the FileGateway.
2. Select 4 at the file type selection menu.
3. Enter the original file name with extension. Include the drive and subdirectory, if needed. The practice file is named *ascii.del*.
4. Enter the database name. Include the drive, if needed.
5. Enter the converted table name or press [ENTER] to use the original file name.

If this is a duplicate table name, either rename the new table, add this data to the existing table, replace existing data, or delete the old table.

6. Make room on the disk if the file is too large for the specified drive. To do this you must exit from the FileGateway.





7. The FileGateway displays:

**Enter the character which separates fields in your input file:**

Enter the character (delimiter) which separates fields in the ASCII file you want to convert.

If you are not sure, press [F10] to display the help menu. Press [5] to display the contents of an ASCII file. Enter the file name (with drive and subdirectory, if the file is not stored on the default directory). When you have identified the delimiter character, select help option 6 to return.

The delimiter character for the sample file is a comma ( , ). Press [ENTER] to specify comma as the delimiter character.

8. Edit the field names, data types, and maximum lengths. Press [ESC] when you are finished editing the field characteristics.
9. Load the data into the database. Select a date format, if needed. If there are no exception records, you are done.
10. Correct and load exceptions.

Edit the highlighted exception fields.

Press [ + ] to load the edited record. The next record, if any, is displayed. If there are no more exception records, the file type selection menu is redisplayed. You are done.

Press [ + ] without editing the exception data. The record is loaded with a null value in the exception field. The next record, if any, is displayed. If there are no more exception records, the file type selection menu is redisplayed. You are done.

Press [D] to delete the displayed exception record. It is not loaded. The next record, if any, is displayed.

Press [ESC] to display the exception loading menu. Options 1 and 2 return to the file type selection menu. Select another file type to process or select option 8 to quit.

## Lotus 1-2-3 Files

### WHAT YOU NEED TO KNOW BEFORE CONVERSION

As a user of Lotus 1-2-3, you are familiar with data manipulation in a worksheet environment. Information is oriented by rows and columns and only a portion of the worksheet is shown at any one time.

You can have a number of spreadsheets stored in different areas on a single Lotus worksheet and, by moving from spreadsheet to spreadsheet, obtain information from each. Similarly, you can retrieve information from any of the tables grouped in a database.

The FileGateway retrieves the data from Lotus 1-2-3 version 1A files and converts it to an R:base format. The structural framework of the worksheet is removed while the data itself is retained. Formulas are not carried over during the conversion, only data and the results of calculations.

Symphony files have a structure similar to Lotus 1-2-3 files so direct conversions may be possible using this option. If the FileGateway cannot convert your Symphony file, convert the file to DIF or ASCII format and use the appropriate FileGateway option to convert.

By working through the sample practice file presented below, you can see how the data in a Lotus 1-2-3 file looks as the FileGateway converts it. This illustrates how your own worksheet files should appear when the FileGateway converts them.

To convert a Lotus worksheet, you need to know certain information about it.

1. Enter the Lotus program or look at a current printout and examine the worksheet you want to convert.
2. If you use labels (field names) on your worksheet, note the row or column on the worksheet which contains the labels.
3. Look at your worksheet and note the two cell coordinates that define the limits of the data you wish to convert using standard Lotus notation—for example, B4..J30. Think of your data as a rectangle and note the letter and number coordinates in the upper-left corner and the lower-right corner.
4. If you do not use labels (field names) in your worksheet, note the orientation of the worksheet (by row or by column).

See "Orientation of Spreadsheets" in this chapter for information on row and column orientation.

**Worksheet and Record Length**

The FileGateway can process records up to 1600 characters long if at least 70 of these characters are expendable characters such as blanks or quotation marks. Because a Lotus spreadsheet can have up to 256 columns and 2,048 rows, it may be necessary to divide it into more than one file. The spreadsheet can be divided either horizontally (as shown in figure 13-11) or vertically if the spreadsheet is row-oriented. The worksheet data is converted by row or column. Each set of data is referred to as a record in the procedures. If any record is longer than 1600 characters (no more than 1530 after FileGateway deletes superfluous quotation marks and blanks), then the record cannot be converted.

Before dividing a spreadsheet into a number of smaller spreadsheets, read "Linking Fields" below.

Figure 13-11 illustrates a Lotus spreadsheet which needs to be split into smaller, separate spreadsheets. In this example, the spreadsheet is divided into three smaller spreadsheets: INCOME, BALANCE, and CASHFLOW. The data coordinates for these spreadsheets are B1..N50, B55..N125, and B130..N160.

	A	B	C	D	E	F	G...	M	N
1	Field name	JAN	FEB	MAR	APR	MAY	JUN	DEC	TOTAL
2									
3	Income Statement								
4	Sales								
.									
50	Net Income								
.									
55	Balance Sheet								
56	Cash								
.									
.	Total Assets								
.	Liabilities								
.									
.	Equity								
125	Total Liabilities								
.									
130	Cash Flow								
.	Cash Sources								
.	Cash Uses								
160	Ending Cash								

Figure 13-11 Lotus 1-2-3 Spreadsheet Example

If you have a Lotus worksheet containing several spreadsheets which are not related, you can avoid linking the tables by not using a common column name. If there is a possible linking column name-such as month, be sure to name each separate spreadsheet month a different name to avoid links. For example, you could assign month labels such as MONTH1, MONTH2, MONTH3 for the separate spreadsheets.

### Linking Fields

When a worksheet is divided into separate spreadsheet files, a link (common characteristic) can be set up in each of the files. In this example the link is row 1 (months). Therefore, before using the FileGateway, each spreadsheet must be altered by repeating the months (row 1) as the first row of the balance sheet and cash flow sections of the spreadsheet.

### File Conversion Editor

In order for R:base to interpret the monthly periods correctly, the data type must be converted from TEXT to DATE and the date format must be defined (for example, MM/DD/YY).

You also need to enter field names having eight or few characters.

### Exception Handling Editor

When the data type for the monthly periods is changed to DATE, the exception handling editor requires conversion of the dates (for example, JAN is changed to 01/01/85). When typing in the date values, it is easiest to use either the first or the last day of the month. An alternative method is to change the date fields to an R:base format using the Lotus program before you begin the conversion process. R:base understands many other date formats but the FileGateway must have more specific information to interpret dates. See the "Data Types and Formats" section for allowable date formats.

### CONVERTING A LOTUS WORKSHEET FILE TO R:BASE FORMAT

Following is a short form of the procedure you must use to convert Lotus files to an R:base format. For details, see the corresponding step in the "General File Conversion Procedures" section.

1. Start the FileGateway.
2. Select 5 at the file type selection menu.
3. Enter the original file name with extension. Include the drive and subdirectory, if needed. The practice file is named *lotus.wks*.
4. Enter the database name. Include the drive, if needed.

5. Enter the converted table name or press [ENTER] to use the original file name.

If this is a duplicate table name, either rename the new table, add this data to the existing table, replace existing data, or delete the old table.

6. Make room on the disk if the file is too large for the specified drive. To do this you must exit from the FileGateway.
7. The FileGateway prompts

**Enter row or column from which to read the field names or [F10] for help:**

If the worksheet does not have labels, press the [ENTER] key alone and continue to step 7A.

If the worksheet has labels for both rows and columns, decide which labels you want to use.

If the worksheet is oriented by row, enter the number of the row in which FileGateway is to find the labels. Continue to step 7B.

If the worksheet is oriented by column, enter the letter of the column in which FileGateway is to find the labels and press the [ENTER] key. Continue to step 7B.

The practice file is oriented by row. Press [5] then press the [ENTER] key.

- 7A. If you do not have labels, the next prompt asks whether your worksheet is oriented by row or column. Respond by entering either *R* (row) or *C* (column).
- 7B. The FileGateway prompts

**Enter the range for the data only in standard  
Lotus notation (ex. B7..M11) or [F10] for help:**

Specify the upper-left and lower-right coordinates of the data area. Do not include the row containing labels in the coordinates. If the worksheet has labels on both row and column, you can include the unselected labels as data or exclude them entirely. Range names may not be used in lieu of actual coordinates.

For the sample file, enter B7..M11 (the standard Lotus coordinate format).

8. Edit the field names, data types, and maximum lengths. Use table 13-15 to properly enter data types in the FileGateway. Press [ESC] when you are finished editing the field characteristics.

Table 13-15 Lotus 1-2-3 Data Formats

Format	Example	R:base Type
General	1000.234	REAL
Fixed	20.123	REAL
Scientific	1.2E+01	REAL
Currency	\$1,000.25	DOLLAR
Comma	1,000.25	REAL
	1,000	INTEGER
Pictograph	+++++	INTEGER (number of characters)*
Percent	15%	REAL
Date	15-Jan-85	DATE
Text	ABCDEF	TEXT

\*Lotus stores an INTEGER number for the pictograph form which is displayed as + + + + +.

9. Load the data into the database. Select a date format, if needed. If there are no exception records, you are done.
10. Correct and load exceptions.

Edit the highlighted exception cells.

Press [+] to load the edited record. The next record, if any, is displayed. If there are no more exception records, the file type selection menu is redisplayed. You are done.

Press [+] without editing the exception data. The record is loaded with a null value in the exception field. The next record, if any, is displayed. If there are no more exception records, the file type selection menu is redisplayed. You are done.

Press [D] to delete the displayed exception record. It is not loaded. The next record, if any, is displayed.

Press [ESC] to display the exception loading menu. Options 1 and 2 return to the file type selection menu. Select another file type to process or select option 8 to quit.

## **dBASE II Files**

### **WHAT YOU NEED TO KNOW BEFORE CONVERSION**

Before converting a dBASE II file, you should be familiar with the file structure. The dBASE II file structure can be displayed or printed using the following procedure.

1. Load the dBASE II program.

2. At the dot prompt, type:

`USE filename`

where *filename* is the name of the dBASE II file.

3. At the dot prompt, type:

`SET PRINT ON`

This allows you to print the file structure. An alternate method is to skip this step and use the [PrtSc] (print screen) key when the structure is displayed.

4. At the dot prompt, type:

`LIST STRUCTURE`

This either displays the structure on the screen (if you did not SET PRINT ON) or prints the structure on your printer (if you did SET PRINT ON).

Use the printed structure when you are editing the converted file for data types and field names.

You can also examine any deleted records in the dBASE II file. The FileGateway gives you the option of converting deleted records. Decide if you want to convert the deleted records or to leave them out of the database file.

While the FileGateway does not directly convert dBASE III records, you can convert your dBASE III files into ASCII delimited format, then convert your dBASE III files using option 4 on the file type selection menu. If your dBASE III files have no more than 32 fields, record lengths less than 1,000, and fewer than 65,535 records, you can also convert them back into dBASE II format. Then, you can use the FileGateway to convert them directly with this option.

If your dBASE II file contains any numeric fields with numbers larger than nine digits but no decimal places, you have a choice of data types when you convert the file. The FileGateway assigns these numbers the DOLLAR data type because: 1) the R:Base INTEGER is limited to nine digits, and 2) the DOLLAR type maintains the accuracy of the number for the full ten digits. The only problem with this is that a dollar sign (\$) is placed at the front of the number. If you change the data type to REAL, the field is treated as an exception. You are required to reenter the number as a six significant digit number in scientific notation.

For example, if you have the number 5,553,334,446 in the dBASE II file, then the number is interpreted by the FileGateway as:

DOLLAR	\$5,553,334,446
--------	-----------------

Figure 13-12 shows three dBASE II files that are linked by common field names. In the *sales* file, it should be noted that more than one entry may contain the same *invoice* number, but the *part:num* and *quantity* information are different.



.LIST STRU				
STRUCTURE FOR FILE:		B:PRODUCT	.DBF	
NUMBER OF RECORDS:		00004		
DATE OF LAST UPDATE:		10/18/84		
PRIMARY USE DATABASE				
FLD	NAME	TYPE	WIDTH	DEC
001	PART:NUM	C	008	
002	PRICE	N	008	002
003	DESCRIPT	C	050	
** TOTAL **			00067	

.LIST STRU				
STRUCTURE FOR FILE:		B:SALES	.DBF	
NUMBER OF RECORDS:		00008		
DATE OF LAST UPDATE:		10/18/84		
PRIMARY USE DATABASE				
FLD	NAME	TYPE	WIDTH	DEC
001	PART:NUM	C	008	
002	QUANTITY	N	004	
003	CUST:ID	N	005	
004	INVOICE	C	008	
** TOTAL **			00026	

.LIST STRU				
STRUCTURE FOR FILE:		B:CUSTOMER	.DBF	
NUMBER OF RECORDS:		00005		
DATE OF LAST UPDATE:		10/18/84		
PRIMARY USE DATABASE				
FLD	NAME	TYPE	WIDTH	DEC
001	CUST:ID	N	005	
002	NAME	C	020	
003	ADDRESS	C	030	
004	CITY	C	008	
** TOTAL **			00064	

common fields

common fields

common fields

common fields

common fields

Figure 13-12 dBASE II File Layout

**CONVERTING A DBASE II FILE TO R:BASE FORMAT**

Following is a short form of the procedure you must use to convert dBASE II files to an R:base format. For details, see the corresponding step in the "General File Conversion Procedures" section.

1. Start the FileGateway.
2. Select 6 at the file type selection menu.
3. Enter the original file name with extension. Include the drive and subdirectory, if needed. The sample file is named *dbaseii.dbf*.
4. Enter the database name. Include the drive, if needed.
5. Enter the converted table name or press [ENTER] to use the original file name.

If this is a duplicate table name, either rename the new table, add this data to the existing table, replace existing data, or delete the old table.

6. Make room on the disk if the file is too large for the specified drive. To do this you must exit from the FileGateway.
7. The FileGateway displays

**Should any deleted records in the dBASE file be transferred Y/N:**

Press [Y] if you want to add records deleted with the DELETE RECORD command in dBASE II. Press [N] to omit deleted records from the conversion. Press [F10] for help.

8. Edit the field names, data types, and maximum lengths. Use table 13-16 to properly enter data types in the FileGateway. Press [ESC] when you are finished editing the field characteristics.

Table 13-16 dBASE II Data Formats

Format	Example	R:base Type
Character	ABCDEF	TEXT
Character	12/15/84	TEXT or DATE
Character	12:01:30	TEXT or TIME
Logical*	F	TEXT
Numeric	231	INTEGER†
Numeric	1234.56	REAL or DOLLAR
Numeric	-.0476	REAL
Numeric	95.6	REAL

\*Logical is a single-character text field to indicate true/false conditions.

†INTEGER up to nine digits. Over nine digits, use DOLLAR. See "What You Need to Know Before Conversion".

9. Load the data into the database. Select a date format, if needed. If there are no exception records, you are done.

10. Correct and load exceptions.

Edit the highlighted exception fields.

Press [ + ] to load the edited record. The next record, if any, is displayed. If there are no more exception records, the file type selection menu is redisplayed. You are done.

Press [ + ] without editing the exception data. The record is loaded with a null value in the exception field. The next record, if any, is displayed. If there are no more exception records, the file type selection menu is redisplayed. You are done.

Press [D] to delete the displayed exception record. It is not loaded. The next record, if any, is displayed.

Press [ESC] to display the exception loading menu. Options 1 and 2 return to the file type selection menu. Select another file type to process or select option 8 to quit.

## **PFS:FILE Files**

### **WHAT YOU NEED TO KNOW BEFORE CONVERSION**

The FileGateway converts data from a file that was built with a PFS:FILE form. The design of the form itself is not important to the FileGateway.

If the file has PFS:FILE attachments—one or more on-screen pages that have been added on to the end of entries of a form—the FileGateway accepts the data from the form but not from the attachment. The information on an attachment is not converted to the new file. However, the FileGateway can convert information added to the form by PFS:REPORTS.

Before you begin to convert a file with the FileGateway, you may want to use PFS:FILE to examine your files. If one or more attachments have significant information that you want to convert, consider redesigning the form, so that the information is carried on the form itself and not on the attachment. See the *PFS:FILE User's Manual* for information on changing forms.

A PFS:FILE file allows up to 32 pages in a form. The FileGateway limits the incoming form size to 1600 characters. If the PFS:FILE form is longer than 1600 characters, the FileGateway cannot convert it. If the PFS:FILE form is between 1530 and 1600 characters and does not contain any non-essential characters (such as quotation marks), then the data past the 1530 character limit is truncated. If your form is longer than 1600 characters then you can split the form into multiple, smaller forms before conversion.

The following example displays a typical PFS:FILE form.

Employee #: M8877		Hired: January, 1984	
Name: Calvin, Curt			
Address: 254 Greentree Lane			
City: Woodville	State: CA	Zip: 93137	
Dept: Manufacturing	Phone Ext: 188		
Job Title: Process Engineer			
Monthly Salary: \$3600			
File: B:STAFF		FORM 8	Page 1
F2-Print Form	F3-Remove Form	F5-Date	F6-Time
		F10-continue	

Figure 13-13 PFS:FILE Form Example

**Linking Files With Common Field Names**

A *linking* field name defines a relationship between files holding different, but related, data on a single subject. For example, if employee salary information is stored in a file named *payroll*, and if employee family information is stored in a file named *personal*, both files should have a common linking field such as *empnum*.

**Field Names**

The FileGateway only accepts field names of eight characters or less. If field names are longer than eight characters, the FileGateway displays the field name and prompts for a new name to be entered.

**File Conversion Editor**

In order for R:base to interpret the *hired* field as a date, the data type must be changed from TEXT to DATE and the date format must be defined (for example, mm/dd/yy).

### Exception Handling Editor

When the data type for the *hired* field is changed to DATE, the exception handling editor requires conversion of the dates (for example, January, 1984 is changed to 01/01/84).

### CONVERTING A PFS:FILE FILE TO R:BASE FORMAT

Following is a short form of the procedure you must use to convert PFS:FILE files to an R:base format. For details, see the corresponding step in the "General File Conversion Procedures" section.

1. Start the FileGateway.
2. Select 7 at the file type selection menu.
3. Enter the original file name with extension. Include the drive and subdirectory, if needed. The sample file is named *pfsfile*.
4. Enter the database name. Include the drive, if needed.
5. Enter the converted table name or press [ENTER] to use the original file name.  
If this is a duplicate table name, either rename the new table, add this data to the existing table, replace existing data, or delete the old table.
6. Make room on the disk if the file is too large for the specified drive. To do this you must exit from the FileGateway.
7. If the FileGateway discovers field names longer than eight characters, it displays:

The above label in your PFS:FILE form is too long to be used as a column name.

Please enter a name of eight characters or less or [F10] for help:

Reenter the column names using eight or less characters for each. If you need help, press [F10].

8. Edit the field names, data types, and maximum lengths. Press [ESC] when you are finished editing the field characteristics.

9. Load the data into the database. Select a date format, if needed. If there are no exception records, you are done.

10. Correct and load exceptions.

Edit the highlighted exception items.

Press [ + ] to load the edited record. The next record, if any, is displayed. If there are no more exception records, the file type selection menu is redisplayed. You are done.

Press [ + ] without editing the exception data. The record is loaded with a null value in the exception field. The next record, if any, is displayed. If there are no more exception records, the file type selection menu is redisplayed. You are done.

Press [D] to delete the displayed exception record. It is not loaded. The next record, if any, is displayed.

Press [ESC] to display the exception loading menu. Options 1 and 2 return to the file type selection menu. Select another file type to process or select option 8 to quit.



## The Application EXPRESS Contents

<b>How to Use This Chapter</b>	14-3
<b>The Application EXPRESS Main Menu</b>	14-3
<b>How to Use the EXPRESS</b>	14-5
Basic Screen Format	14-5
On-Line Help	14-5
Using the[F10]Key	14-5
Using the[F3]Key	14-6
Selecting a Menu Option	14-8
Using the Keyboard	14-8
<b>Defining a New Database</b>	14-10
Naming a Database	14-11
Naming a Table	14-11
Naming a Column	14-12
Selecting a Column Data Type	14-12
<b>Changing an Existing Database Definition</b>	14-13
Adding a Table	14-14
Changing a Table	14-14
Changing Column Names and Data Types	14-15
Adding a Column to a Table	14-15
Removing a Column from a Table	14-17
Removing a Table	14-17
<b>Defining a New Application</b>	14-18
Application Components	14-19
Menu Files	14-19
Screen Files	14-19
Command Files	14-19
R:base Form and Report Files	14-20
Application Files	14-20



Definition Procedure	14-21
Selecting a Database	14-21
Naming the Application	14-21
Naming the Main Menu	14-21
Selecting the Menu Type	14-21
Entering the Main Menu Title	14-21
Entering Text for Main Menu Options	14-22
Selecting a Menu Escape Mechanism	14-22
Defining a Help Screen for the Main Menu	14-22
Selecting Actions for Main Menu Options	14-23
Express Actions	14-24
Storing Application Files	14-27
<b>Changing an Existing Application</b>	14-28
Selecting the Application	14-29
Changing the Application Name	14-29
Changing Menu Text and Options	14-29
Changing the Menu Title	14-29
Adding a Menu Option	14-29
Inserting a Menu Option	14-30
Changing a Menu Option	14-30
Deleting a Menu Option	14-30
Changing the Escape Action	14-30
Changing the Help Action	14-31
Changing an Existing Action	14-32
Removing a Menu from the Application	14-33
<b>Displaying the File Directory</b>	14-34
<b>Exiting the EXPRESS</b>	14-34

## HOW TO USE THIS CHAPTER

This chapter describes how to use the R:base 5000 Application EXPRESS to define and build R:base databases and applications. If the subject of database definition is new to you, be sure to read chapter 2 and work through the tutorial in chapter 3. If you are interested in building an application with the EXPRESS, use the tutorial in chapter 4.

The EXPRESS is a breakthrough program that helps you quickly build applications, even if you have never built an application before. Within minutes, you can create an application that includes a complete database, entry forms, reports, menus, and help screens. To use the EXPRESS, simply follow the instructions that are displayed on the screen. In all cases, you either select a menu option or type in an answer to a question. After you are done, the EXPRESS automatically writes the program code.

## THE APPLICATION EXPRESS MAIN MENU

The Application EXPRESS covers all steps of application building—from defining the database that the application will use to customizing the application with menus, screens, entry forms, and reports.

To get started, select *EXPRESS* from the R:base 5000 main menu or at the DOS prompt enter:

EXPRESS

The following screen shows the EXPRESS main menu. Table 14-1 summarizes the functions of each main menu selection.

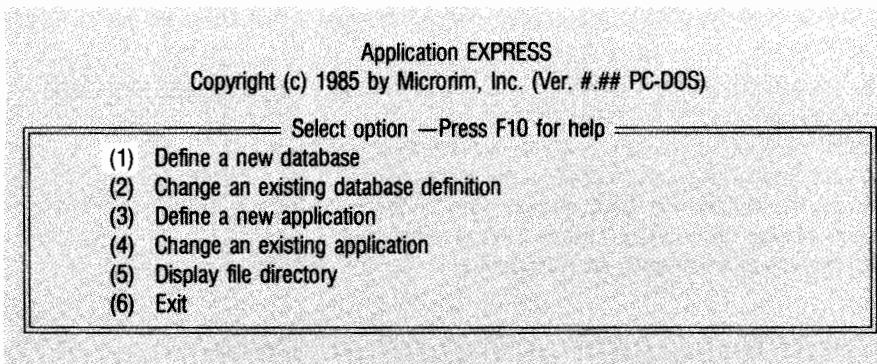


Table 14-1 EXPRESS Main Menu Options and Functions

Option	Purpose
(1) Define a new database	Establish the structure of a new database. Define the database name, table names and structure, and column names, data types, and lengths.
(2) Change an existing database definition	Add and delete tables or columns, rename tables or columns, and change column data types and lengths.
(3) Define a new application	Build a new application for a database. Define the menu names, types, titles, options, and actions.
(4) Change an existing application	Add, delete, or change menu options or actions.
(5) Display file directory	Display information about the disk files on the current directory.
(6) Exit	Leave the EXPRESS to return to the R:base 5000 main menu or to the DOS prompt.

The EXPRESS offers these advantages:

- Development time is greatly reduced. You can start using your database or application more quickly.
- You do not need to learn a command language to transform your design into a database or application. The EXPRESS walks you through the process.
- Databases and applications are easy to change.
- You can use the EXPRESS to join existing command files (macros) with a new application.

The EXPRESS, while a powerful application tool, does have certain limitations. These are:

- Maximum of 40 columns may be defined for a database
- Maximum of three columns may be sorted on
- Conditional lists (WHERE clauses) are limited to a single condition
- Reports are limited to a width of 80 characters
- Entry forms are limited to 20 columns

## HOW TO USE THE EXPRESS

The EXPRESS has several features that you should familiarize yourself with. They are listed below and explained in the following paragraphs.

- Basic screen format
- On-line help
- Selecting a menu option
- Using the keyboard

### Basic Screen Format

The EXPRESS divides the display screen into three functional areas. The top area of the screen displays a prompt or menu that describes the task at hand. For example, there may be a one-line prompt such as *Enter your database name.*

The middle area of the screen is the work area. You may be prompted to fill in the blanks, choose a menu option, or respond to a question. The work area cursor always orients you visually to the place to enter your next input and it is always obvious whether you should fill in a blank, respond to a prompt, or make a menu choice.

The bottom area of the screen displays the name of the database that you are working on, the task you are performing, and, if you are working with a table, a column number. For example, when you are creating a table named *custlist* in a database named *compuco*, a typical status line looks like this:

[F1] Insert	[F2] Delete	[F3] Review	[F5] Reset value	[F10] Help
Database compuco	---	Defining table custlist	---	Column 1

### On-Line Help

#### USING THE [F10] KEY

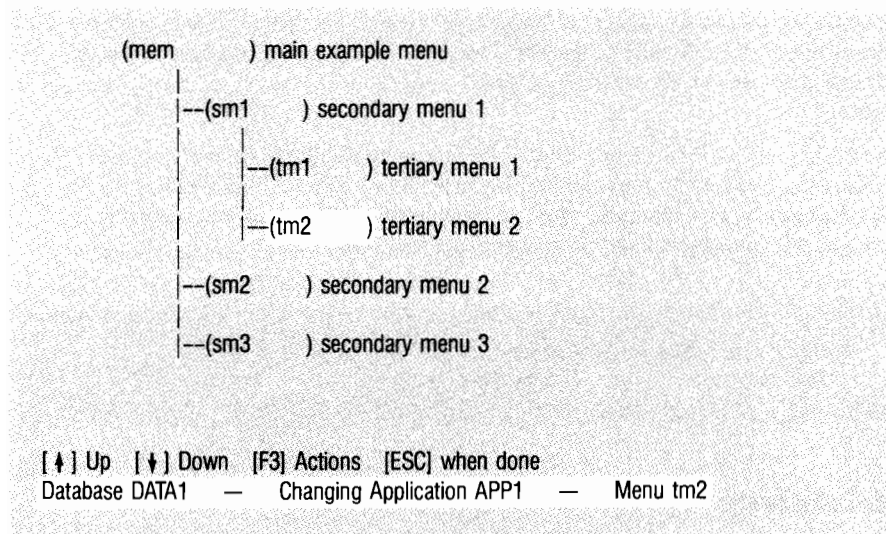
Press [F10] to display on-line help text for the task you are currently working on. Help information is displayed in a window in the screen work area, leaving all but five lines of the current screen display intact. In most cases, help is placed on the screen so it does not overlay any of your work area display. This way, you can read the text and compare it with your on-screen task. If there is more than a single help screen, a message instructs you to press [F10] for the additional text. When you no longer require the help text, press any key to return to your immediate task.

**USING THE [F3] KEY**

When you are defining or redefining a database, press [F3] to display all the defined tables and columns in the open database.

When you are defining or redefining an application, press [F3] to display a visual representation of the application menu tree. The menu name, title, and level are all shown.

For example, suppose you have an application *APP1* with three levels of menus defined for a database named *data1*. The main menu is named *mem* and titled *main example menu*. *Main example menu* has three options, each of which process another menu. The titles of the menus are *secondary menus 1, 2, and 3*. In addition, *secondary menu 1* has two options, and both of them process a menu. These menus, which are three levels deep, are titled *tertiary menus 1 and 2*. If you press [F3] when you are modifying *tertiary menu 2*, your screen looks like this:



The name of the menu you were working on when you pressed [F3] is highlighted in the work area and displayed at the bottom of the screen. You can move the cursor from menu to menu in the menu tree with the up-arrow and down-arrow keys.

If you want to display the options on a specific menu, move the cursor to that menu and press [F3] again. For example, if you want to list the options on the *secondary menu 1* menu, move the cursor to *sm1* and press [F3]. The display looks similar to this:

secondary menu 1	
ADD CHANGE REMOVE INQUIRE	
Menu selection 1	Process menu: tm1
Menu selection 2	Process menu: tm2
Menu selection 3	none
Menu selection 4	Select table: salesrep
Press any key to continue	
Database DATA1 — Changing Application APP1 — Menu submenu1	

The preceding screen shows:

- The actions *Select* and *Menu* are used to implement the options in the menu.
- The first and second menu options lead to the display of a submenu.
- No action is defined for the *REMOVE* option.
- There is one action defined for the *INQUIRE* option.

To return from this screen to the menu tree, press any key. When you are finished browsing through the menu tree for information, press [ESC] to return to the application-building task you were doing when you stopped to look at the structure.

## Selecting a Menu Option

There are two ways to select an option from a menu displayed by the EXPRESS:

- Move the cursor to the desired option with one of the following keys and press [ENTER]:

Space bar -----	Move right (or down) 1 item
Right-arrow -----	Move right (or down) 1 item
Left-arrow -----	Move left (or up) 1 item
Up-arrow -----	Move up 1 line
Down-arrow -----	Move down 1 line
Tab -----	Move to last item on current line (horizontal menus)
Shift and Tab -----	Move to first item on current line (horizontal menus)
Home -----	Move to first item
End -----	Move to last item

- Type the first letter or number of the option you want. This causes the cursor to move to that option and execute it. If a menu has more than one option that starts with the same letter, press [ENTER] after the option you want is highlighted.

In both cases, the cursor movement is circular. For example, if you move the cursor to the last item in the list with the right-arrow key, and press the right-arrow key again, the cursor moves back to the start of the list.

## Using the Keyboard

Table 14-2 contains a complete list of the keyboard keys available for use with the EXPRESS. The specific meaning of each editing key depends upon whether you are editing a table, a vertical menu (VMEN), or a horizontal menu (HMEN).

Table 14-2 The EXPRESS Keyboard Functions

Key	Used By	Definition
[PgDn]	ALL	Moves cursor from title to menu or table
[PgUp]	ALL	Moves cursor from menu or table to title
[→]	TABLE,HMEN	Moves cursor one column or option to the right
	VMEN	Moves cursor one row down
[←]	TABLE,HMEN	Moves cursor one column or option to the left
	VMEN	Moves cursor one row up
[Home]	ALL	Moves cursor to first column or option
[End]	ALL	Moves cursor one position past last column or option
[F1]	ALL	Inserts blank column or option at current cursor position
[F2]	ALL	Deletes column or option at current cursor position
[→]	ALL	Moves cursor one character right
[←]	ALL	Moves cursor one character left
[↑]	VMEN	Moves cursor one row up
[↓]	VMEN	Moves cursor one row down
[Ins]	ALL	Inserts blank character at current cursor position
[Del]	ALL	Deletes character at the current cursor position
[← ]	ALL	Changes character one space left of current cursor position to a blank
[Ctrl] [→]	ALL	Moves cursor one space past last character in column or option
[Ctrl] [←]	ALL	Moves cursor to start of reverse video field
[ENTER]	TABLE	Enters table or column name and moves cursor one column right
	HMEN,VMEN	Enters menu title or option text. If option text, moves cursor to the next option
[ESC]	TABLE	Enters table or column name and quits editing table
	HMEN,VMEN	Enters menu title or option text and quits editing menu
[F5]	ALL	Restores text in reverse video field to its original value. Not operational after video field highlights different column or option



## DEFINING A NEW DATABASE

Defining a database is easy with the Application EXPRESS. In most cases, you will prefer using it to the DEFINE mode in R:base. For some tasks, such as assigning passwords, you will want to use the DEFINE mode or database-structure commands in R:base. Table 14-3 shows the definition tasks that you can accomplish with each method.

Table 14-3 Database Definition and Redefinition: The EXPRESS Compared with DEFINE and R:base Commands

Task	The EXPRESS	DEFINE Mode	R:base Command
Name a database	XX		DEFINE
Define a new table	XX	TABLES	
Define a new column	XX	COLUMNS	
Define an owner password		OWNER	
Define a table password		PASSWORDS	
Define data entry rules		RULES	
Redefine a column data type for a column used in one table	XX		CHANGE COLUMN
Add a column to a table	XX		EXPAND
Rename a column	XX		RENAME COLUMN
Rename a table	XX		RENAME TABLE
Remove a column from a table	XX		REMOVE COLUMN
Remove a table	XX		REMOVE TABLE
Rename the owner password		OWNER	RENAME OWNER
Build a key for a new column		COLUMNS	
Build a key for an existing column			BUILD KEY
Delete a key for a column			DELETE KEY

## Naming a Database

Choose option 1 from the EXPRESS main menu to define a new database. The EXPRESS responds with the following message.

Enter your database name (1-7 characters)

A database name can be from one to seven characters. The name cannot begin with a number or special character and it cannot contain blank spaces. The database is built on the default drive and current directory. After you enter the database name, a blank table is displayed and the status line at the bottom of the screen displays the database name you entered.

## Naming a Table

Your next step is to define a table name. Table names can be from one to eight characters in length. The name should not begin with a number or special character and it cannot contain blank spaces. Assign the table a name that describes the table function. For example, in the *compuso* database, the table that contains information about sales representatives is named *salesrep*.

Enter the name for this table

salesrep


Database compuco

When you press [ENTER] at the end of the table name, the cursor moves to the top of the first column on the table and the EXPRESS responds with the following message:

Enter or change the column names—[ESC] when done

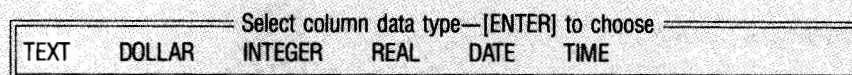
Also, the status line at the bottom of the screen changes to reflect the new status of your general database definition task.

## Naming a Column

A column name can be from one to eight characters in length. The name should not begin with a number or special character and it cannot contain blank spaces. Again, it is a good idea to use a name that describes the column function. For example, you may want to name a column of employee identification numbers *empid*.

### SELECTING A COLUMN DATA TYPE

After you enter a column name, you need to select the column data type. The EXPRESS displays the following menu:



Select column data type—[ENTER] to choose					
TEXT	DOLLAR	INTEGER	REAL	DATE	TIME

Press the first letter of the type you want, or move the cursor across the menu with the space bar, right-arrow, or tab keys. Then press [ENTER].

Select TEXT for data that contains text, special characters, or blanks. TEXT values cannot be used for arithmetic functions.

If you select TEXT, you have to enter a text length, which is the largest number of characters that may be contained in a value entered in the column. Enter a length between 1 and 1500. The EXPRESS uses a length of eight characters by default if you do not enter a length.

Select DOLLAR for currency values. The range of values is from zero to plus or minus \$99,999,999,999,999.99. All arithmetic operations (+, -, ×, /, and %) and COMPUTE command functions (SUM, AVE, MIN, MAX, and COUNT) may be done with DOLLAR values.

Select INTEGER for whole numbers (numbers without a decimal point). The range of values is from zero to plus or minus 999999999. INTEGER values can be used in all arithmetic operations (+, -, ×, /, %) and COMPUTE command functions (SUM, AVE, MIN, MAX, and COUNT).

Select REAL for numbers that require a decimal point. The range of values is from zero to plus or minus  $9 \times 10^{\pm 37}$ . REAL values can be used in all arithmetic operations (+, -, ×, /, and %) and COMPUTE command functions (SUM, AVE, MIN, MAX, and COUNT).

Select DATE for values that represent a month, day, and year. For example, 06/28/85. Addition and subtraction may be done with a DATE value and an integer. The difference in days between two DATE values can be calculated. The format for dates first lists the month, then the day, and then the year.

Select TIME for values that represent clock times as hours, minutes, and seconds. For example, 06:10:00 and 18:10:00. Addition and subtraction may be done with TIME values.

After you have named the column and selected its data type, the cursor moves to the second column in the table. Repeat the process described above until all the columns for the table are defined.

The EXPRESS displays seven columns on the screen at one time. When you define eight or more columns, the display shifts to the right and the columns that you first defined are not displayed. The status line at the bottom of the screen, however, always indicates the column number you are currently defining. If you need to insert a column between two columns use [F1]. To delete a column use [F2]. For complete information on the use of these function keys, see the "Change an Existing Database Definition" section of this chapter.

After you enter the last column in the table press [ESC]. The EXPRESS displays a menu that gives you the option to add a new table to the database, change an existing table, remove a table from the database, or exit to the main menu.

## CHANGING AN EXISTING DATABASE DEFINITION

Select option 2 from the EXPRESS main menu to accomplish the following tasks:

- Define a new table for the database
- Change a table name
- Change a column name
- Change a column data type or length
- Add a column to a table
- Delete a column from a table
- Remove a table from the database

You can change the structure of a database table with the EXPRESS only if it contains no data. For tables that contain data, use R:base commands to make changes.

When you select option 2, the EXPRESS displays the names of all the existing databases in the current directory. Move the cursor to the database you want to change and press [ENTER]. The EXPRESS displays the database name on the status line and the following menu:

Select table option	
(1)	Add a new table to the database
(2)	Change an existing table
(3)	Remove a table from the database
(4)	Exit to the main menu

Press the number of the option you want, or move the cursor to the option and press [ENTER].

## Adding a Table

Select option 1 from the menu to add a table to the database. Enter the table name and define the column names and types. If you need help with any of these steps, follow the instructions in the “Defining a New Database” section of this chapter.

## Changing a Table

Select option 2 from the menu to:

- Change a table name
- Change a column name
- Change a column data type or length
- Add a column to a table
- Delete a column from a table

The EXPRESS displays a list of the tables in the database. Move the cursor to the table you want to change and press [ENTER]. The EXPRESS displays the table and highlights the table name. Press [ENTER] to use the existing name, or type a new name and then press [ENTER]. For example, suppose you want to change the name of the *transx* table to *transact*. When the *transx* table definition first appears in the screen work area, the cursor highlights the table name. Type the new table name and press [ENTER]. You now can change column names, data types, and lengths, or add or remove columns. If you want to return to the table name, press [PgUp] at any time.

## CHANGING COLUMN NAMES AND DATA TYPES

The following instructions explain how to change a column name, data type, and length. When you rename a column, you cannot assign it the same name as another column in the same table. In addition, if you enter a column name that exists in another table in the database, the data type and length are automatically assigned. If you want to change the data type of a column that exists in more than one table in the database, you must first rename the column.

Move the cursor to the column you want to change. If you want to change the name, type the new name and press [ENTER]. If you want to change only the data type/length, press [ENTER].

After you press [ENTER], the EXPRESS displays the six data types with the current data type highlighted. Press [ENTER] if you do not want to change the type. Otherwise, press the first letter of the type you want, or move the cursor across the menu with the space bar, right-arrow, or tab keys. Then press [ENTER]. A complete explanation of the data types is in the “Defining a New Database” section of this chapter.

As an example of a column change, suppose you want to change the *location* column in the *product* table to a new name, *aisle*, and you also want to change the length from eight to 12 characters. Move the cursor to the *location* column, type in the new name, *aisle*, and press [ENTER]. Choose the option *TEXT*, and enter the new length of 12. Move the cursor to another column, or press [ESC] if the table does not require further change.

## ADDING A COLUMN TO A TABLE

The order of the columns in a table makes no difference to system performance. You may, however, want to change the location of a column to clarify the screen presentation of the table.

If you want to add a column to the end of a table, press [End] to move the cursor to the new column. Enter the column name and data type. If you enter the name of a column that exists elsewhere in the database, the data type and length are automatically filled in.

If you want to add a column in the middle of a table, move the cursor to the column in the table where you want to locate the new column and press [F1]. The EXPRESS moves the existing columns right and inserts a blank column. Enter the column name and data type. If you enter the name of a column that exists elsewhere in the database, the data type and length are automatically filled in.

In the COMPUCO example, suppose you want to add a new column, *spares*, between the columns *prodname* and *onhand* in the table *product*. The data type is TEXT, and the length is 4. The existing table is shown in the following screen.

product						
prodid	proddesc	prodname	onhand	cost	listprce	location
TEXT 6	TEXT 31	TEXT 35	INTEGER	DOLLAR	DOLLAR	TEXT 8

Move the column cursor to the *onhand* column and press [F1]. The table now appears as shown in the following screen.

product						
prodid	proddesc	prodname		onhand	cost	listprce
TEXT 6	TEXT 31	TEXT 35		INTEGER	DOLLAR	DOLLAR

Enter the new column name, *spares*, and press [ENTER]. Choose *TEXT* from the data type menu, and enter a length of 4. The following screen shows the final result.

product						
prodid	proddesc	prodname	spares	onhand	cost	listprce
TEXT 6	TEXT 31	TEXT 35	TEXT 4	INTEGER	DOLLAR	DOLLAR

### REMOVING A COLUMN FROM A TABLE

You can also remove a column from a table. Move the cursor to the heading of the column you want to delete and press [F2]. The EXPRESS removes the column and the space it occupied from the display. If the column exists elsewhere in the database, it is only removed from the table currently displayed.

For example, if you want to remove the column *spares* from the *product* table that is shown in the previous screen, move the cursor to the *spares* heading and press [F2]. The column is removed, and the display shows the *prodname* and *onhand* columns immediately adjacent to each other.

### Removing a Table

Select option 3 from the menu to remove a table from the database. The EXPRESS displays a list of the tables in the database. Use the space bar, right-arrow, or tab key to move the cursor to the table you want to remove and press [ENTER]. The EXPRESS displays a message that shows the table you selected and asks you to confirm your decision. Press [ENTER] to remove the table, or press [ESC] to abort the command.



## DEFINING A NEW APPLICATION

An R:base application consists of two major parts: a database and an application program. To make your application perform as well as possible, plan ahead. Sketch a model of your program on paper. Refine the model. Then, from your plan, use the EXPRESS to define the application on your computer. Chapter 3 explains the concepts involved in database design, and the remainder of this section pertains to application development. In this chapter, the sample database *compuco* is used to illustrate examples. Figure 14-1 shows an application design sketch for the *compuco* database.

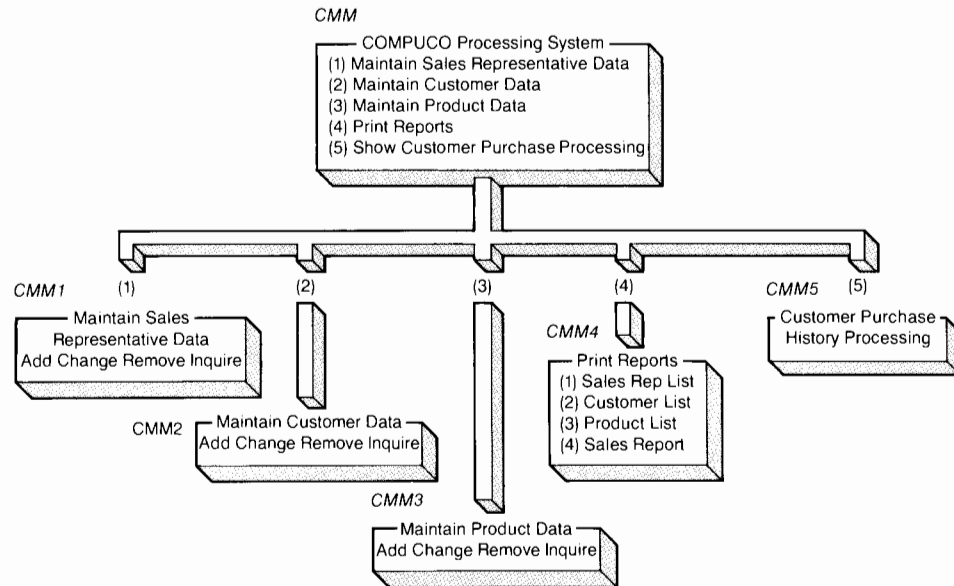


Figure 14-1 COMPUCO Application Design Sketch

After you are comfortable with the application design, build it with the EXPRESS. Test the application, or parts of it, using a database that contains a dozen or so representative rows of data. Then redefine the database, the application, or both until you are completely satisfied with the final result.

## Application Components

An application is used to create a link between a computer and a computer user so that the user can efficiently accomplish an objective without entering line after line of commands. This link is composed of three types of files: menu files, screen files, and command files. An EXPRESS application file combines the information from all three file types to execute a series of commands that accomplish a complete task.

### MENU FILES

Menu files contain the information and structure used by the R:base CHOOSE command to display a menu on the screen. A menu is a computer screen display that informs the user of available options. When the user makes a choice, the application performs the option and displays another menu. An application that consists entirely of menu choices is called a menu-driven application.

The EXPRESS offers two types of menu formats, vertical and horizontal. Vertical menus display the menu choices in a column format, listing one option on a line. Horizontal menus display the choices in a row format, listing several options on one line.

### SCREEN FILES

Screen files contain the information and structure used by the R:base DISPLAY and TYPE commands to show messages on the computer screen.

The EXPRESS uses screen files to make help screens. For example, they can present definitions of technical terms that appear in the menu choices, instructions on how to make a choice, or implications of making a certain menu choice. In EXPRESS-built applications, one help screen may be associated with each menu. Not all menus need to have a help screen; the information in the menu title and in the text that describes each choice may be sufficient. Help screens are only displayed if the user signals for help by pressing [F10].

### COMMAND FILES

An R:base command file is a list of commands that is executed (or run) to manipulate an R:base database. Each command line contains information about the execution of a specific R:base command. A command file is started either by a user choosing a menu option or by a command line in another command file.

There are three types of command files available to the developer:

- EXPRESS actions
- Predefined files (macros)
- Custom



You can use EXPRESS actions without having to do any programming. You need to design custom command files when the application you are building requires a sequence of commands not packaged as an EXPRESS action or as an existing macro. The functions available as EXPRESS actions are listed in table 14-4. See chapter 15 for information on programming command files.

Table 14-4 EXPRESS Action Command Files

Action	Function
Load	Define a table form and use it to load rows into a table
Edit	Define a table form and use it to edit values in selected table rows
Delete	Delete selected rows from a table
Browse	View selected rows from a table in sorted order, editing values when desired
Select	Display selected rows in sorted order
Print	Define a report and use it to print information from a table
Custom	Define a custom command file
Macro	Use existing command file code
Menu	Define a new submenu
Exit	Go to the application main menu or leave the application

### R:BASE FORM AND REPORT FILES

In addition to forms and reports that you create with the EXPRESS, you can also include R:base custom data entry forms and reports in your command files. You can define data entry forms with the R:base FORMS command and reports with the REPORTS command. The *forms* table contains the information and structure used by the EXPRESS *Load* and *Edit* actions to enter and edit information in a table. The *reports* table contains the information and structure used by the EXPRESS *Print* action.

### APPLICATION FILES

After you define the parts of your application, you need to join them together in an application file. The EXPRESS automatically does this for you. It can build a file that contains up to 42 blocks. There are three types of blocks:

- Command blocks, which are command files inside an application file
- Menu blocks, which are menu files inside an application file
- Screen blocks, which are display files inside an application file

## Definition Procedure

Be sure the default drive and current directory contain the database associated with your application. Select option 3 from the EXPRESS main menu to define a new application for an existing database.

### SELECTING A DATABASE

The EXPRESS displays the database names on the default drive and directory. If the database associated with your application is not listed, exit the EXPRESS and establish the default drive and current directory where the database exists.

Select the database you want from the list. The EXPRESS then asks you for an application name.

### NAMING THE APPLICATION

Application names can be from one to eight characters long. The name cannot contain blank spaces or special characters. Assign an application name that describes the application function. For example, if this is the first application for the *compuco* database, you might want to name the application *compuco1*. Enter a name and press [ENTER]. The EXPRESS prompts you for a main menu name.

### NAMING THE MAIN MENU

Menu names contain from one to eight characters. Assign a menu name that implies this is the main menu for the application. For example, you may want to abbreviate the *compuco1* main menu as *cmm1*. After you enter the main menu name, the EXPRESS wants to know which menu format to use for the menu.

### SELECTING THE MENU TYPE

The EXPRESS enables you to build two types of standard menus for your application: vertical and horizontal. Vertical menus list the menu options in a column, one option per line. Each line can contain an option description up to 60 characters in length. Horizontal menus list the menu options across a row. The maximum length for each option is ten characters.

It is usually a good idea to use a vertical menu for the main menu of an application. Move the cursor to the option you want and press [ENTER].

### ENTERING THE MAIN MENU TITLE

Enter the main menu title. You may enter up to 60 characters into the reverse video field. Assign a title that helps users orient themselves in the application. After you enter the title, the EXPRESS automatically centers the title.

**ENTERING TEXT FOR MAIN MENU OPTIONS**

Press [ENTER] after you have entered all the characters for one option. The EXPRESS displays a reverse video field at the next option location.

After you enter all the main menu options, press [ESC]. The EXPRESS moves your main menu from the screen work area to the task area at the top of the screen. The task area cursor highlights option 1.

**SELECTING A MENU ESCAPE MECHANISM**

Select the method the user must use to escape from the main menu. There are two methods from which to choose: pressing [ESC], or defining an *Exit* option for the menu.

The EXPRESS displays the following message:

```
Is ESC to be used to exit this menu?  Yes  No
```

If you included an *Exit* option in the application main menu, select *No* unless you want to include [ESC] as an alternate option.

**DEFINING A HELP SCREEN FOR THE MAIN MENU**

Next, the EXPRESS asks you if you want to build a help screen for the menu. The following message is displayed:

```
Is HELP to be defined for this menu?  Yes  No
```

You can use the EXPRESS to build one help screen for each menu in an application. Use help screen text to define technical terms that appear in the menu, give instructions on how to make a menu choice, or list the result of each choice. If you want to define a help screen, select *Yes*. The EXPRESS screen is replaced by the RBEDIT screen, and you can select up to 24 lines of text. Additional information on the editing keys available with RBEDIT is in chapter 15.

If you do not want to define on-line help, select *No*. You are finished defining the application main menu.

## SELECTING ACTIONS FOR MAIN MENU OPTIONS

Now that your main menu is built, you can define the actions for each of the main menu options. If you do not want to define an action for the current pick, press [ESC] to skip to the next action. The EXPRESS prompts you with the following screen.

```

First Compuco Application Main Menu
(1) Maintain Sales Representative Data
(2) Maintain Customer Data
(3) Maintain Product Data
(4) Print Reports
(5) Customer Purchase History Processing
  
```

The actions for menu selection 1 will be defined.

```

Select action for this menu selection
Load Edit Delete Browse Select Print Custom Macro Menu Exit
  
```

```

[F3] Review      [F5] Reset value    [F10] Help
Database COMPUCO - Defining Application compuco1 - Menu CMM
  
```

The horizontal menu in the work area shows all the actions you can define—one at a time—for the option:

- Select *Load* to enter data into a table.
- Select *Edit* to modify the information in a table.
- Select *Delete* to remove rows of data from tables.
- Select *Browse* to view and edit the contents of a table.
- Select *Select* to display specific information in a table.
- Select *Print* to print a report.
- Select *Custom* to build a custom command file.
- Select *Macro* to select an existing command file.
- Select *Menu* to build a submenu.
- Select *Exit* to add an exit option to the menu.

**Express Actions**

The six EXPRESS actions—*Load*, *Edit*, *Delete*, *Browse*, *Select*, and *Print*—are the fundamental table actions. When you select an action, the EXPRESS leads you through each step of the definition process.

**Entering Data into a Table.** *Load* is the EXPRESS action that adds rows to a table using a data entry form.

General Notes:

- Select a table from the displayed list. The EXPRESS asks you for the name of the data entry form you want to use.
- If the database you are working with already has at least one defined form, EXPRESS lists the existing form names and adds the choice (*NEW*). If no forms exist, EXPRESS asks you to assign a name to the form. The name can be from one to eight characters in length, and cannot contain blank spaces or special characters. Note that a form can have the same name as a table.
- If the desired form for the table exists, select it. You are finished defining the *Load* action for the application menu option.
- If you have to define a new form, the EXPRESS prompts for a form name, title, and prompt text. You can enter up to 25 characters of descriptive text for each prompt.
- The EXPRESS displays a list of the columns in the table, plus the option (*ALL*). Select (*ALL*) to load all the columns on the form in the same order as they are listed in the table (from left to right). The maximum number of columns that you can load is 20. If the table contains more than 20 columns, you cannot select (*ALL*). Choose 20 or less columns from the column list.
- If you want the order of the prompts on the form to be different from the order of the columns in the table or you do not want the user to be prompted for all the columns, use the cursor to select column names individually.
- After an item is selected from the column list, it is displayed in the area below and the (*ALL*) option inside the menu changes to (*RESET*). If you want to reset the column selections to the original list, select the (*RESET*) option.

**Modifying Data in a Table.** *Edit* is the EXPRESS action that enables an application user to change row values in a table.

General Notes:

- Select a table from the displayed list. The EXPRESS asks you for the name of the form you want to use.
- If the database you are working with already has at least one defined form, EXPRESS lists the existing form names and adds the choice (*NEW*). If no forms exist, EXPRESS asks you to assign a name to the form. The name can be from one to eight characters in length, and cannot contain blank spaces or special characters. Note that a form can have the same name as a table.
- You may want to define a new form for editing even if a form is defined for loading. You can define a form that does not prompt the user for specific row values that are confidential or too critical to risk incorrect edits.
- Select a sorting order, based on up to three columns.
- Specify a condition that rows must meet to be displayed for editing. The general form of the condition has two or three parts:

column name, condition  
or  
column name, condition, comparison value

When the column list is displayed, choose the column you want to use with the comparison condition. Enter the *comparison value* when you develop the application or prompt the user to fill in a value when the application is run.

The following conditions are available:

*EQ* means equal. Only rows that have a value equal to the entered value are displayed.

*NE* means not equal. Only rows that have a value unequal to the entered value are displayed.

*GT* means greater than. Only rows that have a value greater than the entered value are displayed.

*GE* means greater than or equal to.

*LT* means less than. Only rows that have a value less than the entered value are displayed.

*LE* means less than or equal to.

*CONTAINS* means that only rows that have a value that contains the same a string of characters as the entered value are displayed.

*EXISTS* means that only rows that have a non-null value are displayed. No comparison value is needed.

*FAILS* means that only rows that have a null value are displayed. No comparison value is needed.

The EXPRESS also asks whether the comparison criteria will be entered at the time the option is selected and, if so, what the entry prompt will be.



**Deleting Rows from a Table.** *Delete* is the EXPRESS action that enables you to delete rows from a table.

General Notes:

- Select a table from the displayed list.
- Specify a condition that selects rows to delete by following the same procedure as specified for the *Edit* action.

**Browsing Through a Table.** *Browse* is the EXPRESS action that enables you to display a table on the screen and view all the columns and rows using the cursor control keys. The user may edit values in the table as well as look at them.

General Notes:

- Select a table from the displayed list.
- Select columns to browse through (up to 10), or (ALL).
- Select a sorting order, based on up to three columns.
- Specify a condition that selects rows to display by following the same procedure as specified for the *Edit* action.

**Selecting Rows from a Table.** *Select* is the EXPRESS action that enables an application user to display rows from a table.

General Notes:

- Select a table from the displayed list.
- Select columns to display (up to 10), or (ALL).
- Select a sorting order, based on up to three columns.
- Specify a condition that selects rows to display by following the same procedure as specified for the *Edit* action.

**Printing a Report.** *Print* is the EXPRESS action that enables you to print a report from a table.

General Notes:

- Select a table from the displayed list.
- Select the report form to use, or define a new one.
- If the desired form for the report exists, select it. You are finished defining the *Print* action for the application menu option.
- If you have to define a new report, the EXPRESS prompts for a report name, title, and headings. You can enter up to 25 characters of descriptive text for report headings.
- Select row-or column-wise orientation. A column-wise report prints the table data in columns.
- Select an existing report definition, or define a new one.
- Select the columns to print (up to 20) or (ALL). The total printing width may not exceed 132 characters.

- The width of each column is determined by the column data type. The width of each column is shown in reverse video when you define the column headings.
- On the third line of the screen is a reverse video field into which you can enter the report title. The title can be up to 78 characters long.
- Select a sorting order, based on up to three columns.
- Specify a condition that selects rows to print by following the same procedure as specified for the *Edit* action.

**Building Custom Command Files.** *Custom* is the EXPRESS action that enables you to add your own command code to the EXPRESS application. When you select *Custom*, the RBEDIT screen is displayed. Enter your own command file code. The code you write is added to the application file as a separate command block. The block is called as a subroutine whenever the menu item defined as *Custom* is selected.

**Building Macros.** *Macro* is the EXPRESS action that enables you to use existing command file code in your application. These existing command files, if designed for use in more than one application, are called *macros*. Chapter 15 describes how to design and build your own command files to use as macros. EXPRESS allows a macro to be used only once in the same application.

**Building Menus.** *Menu* is the EXPRESS action that enables you to build submenus up to three levels deep.

**Exiting Main Menu Actions.** *Exit* is the EXPRESS action that directs the application to leave the current menu and go to the next higher level menu in the menu tree. If *Exit* is an option on the main menu, you are returned to an R > prompt or to DOS depending on the commands in the *rbase.dat* file.

## STORING APPLICATION FILES

When you finish defining your application, the EXPRESS stores your application files. As the EXPRESS stores the file, it displays each command line on the screen while the application file is being compiled and makes three files on disk. For example, when *compucol* is saved, the application files created are *compucol.api* (pointer file), *compucol.apx* (compiled file), and *compucol.app* (ASCII file). If you change the application file at a later time with a text editor instead of the EXPRESS, the ASCII file—the file with the *.app* extension—may be converted to a binary file with RCOMPILE. If the ASCII application file is modified outside of the EXPRESS, you cannot use the EXPRESS to make any further changes. You may modify any *Custom* code you have written for the application using RBEDIT available within the EXPRESS.

After the application is stored, the EXPRESS directs you to press any key to continue. When you press a key, the EXPRESS displays the following screen:

Do you want a start-up file for this application? ([F10] for help) Yes No

This option allows you to build a start-up file for the application named *rbase.dat*. An *rbase.dat* file is a command file that automatically executes the application each time R:base is started. If you are using a hard disk system and want to define an *rbase.dat* file, select *Yes*. Otherwise, select *No*.

If you do not build an *rbase.dat* file, you will need to enter the R:base RUN command at an R> prompt to start your application. For example, to start the *compuco1* application, at the R> prompt type:

```
RUN compuco1 IN compuco1.apx
```

Additional information on using the RUN command to execute command files is in chapter 15.

If you do build an *rbase.dat* file, it contains the RUN command and an EXIT command. The EXIT command is used to exit from R:base when the operator leaves the application. To make sure an *rbase.dat* file does not interfere with other R:base operations, be sure to keep each custom application's *rbase.dat* file on the same directory as the application files. Then, when you want to execute a different application or load R:base to work, make sure the proper directory is current. Alternatively, you can rename the *rbase.dat* file for an application when you do not want to use it.

## CHANGING AN EXISTING APPLICATION

Select option 4 from the EXPRESS main menu to do any of the following tasks:

- Change the application name
- Remove a menu from the menu tree
- Change a menu title
- Change the text of a menu option
- Delete a menu option (and it's associated actions)
- Add a menu option
- Insert a menu option between existing options
- Change the menu escape mechanism
- Change the help screen associated with the menu
- Change an action for any menu option
- Add an action to the sequence of actions associated with any menu option, either before or after an existing option

If you have added an OWNER password to the database for an existing application, you will not be able to modify the application. First, you must use the R:base CHANGE command to remove the password as follows:

```
CHANGE OWNER password TO NONE
```

## Selecting the Application

When you select option 4 from the EXPRESS main menu, a list of the applications on the current directory is displayed. Select the application you want to change.

## Changing the Application Name

After you select the application, the EXPRESS displays the following prompt.

```
Do you want to change the application name:  Yes  No
```

If you select *Yes*, you are prompted for a new application name. If you select *No*, or after you enter a new application name, the EXPRESS reads the existing application files from disk. Then it displays the application change menu:

```
----- Select application change option -----
(1) Change menu text and options
(2) Remove a menu from the menu tree
```

## Changing Menu Text and Options

Select option 1 from the application change option menu to change menu text and options. The EXPRESS displays a list of the menus in the application. Move the cursor to the menu you want to change and press [ENTER].

### CHANGING THE MENU TITLE

The menu you want to change is displayed. The menu title is left-justified in a reverse video field and you are prompted for a new menu name. To keep the same title, press [ENTER]. Otherwise, type in a new title and press [ENTER]. In either case, after [ENTER] is pressed, the EXPRESS automatically centers the title and the cursor moves to the first menu option.

### ADDING A MENU OPTION

To add a menu option to the end of a menu, press [End]. The cursor moves to the last column or row and you can enter the option text.

### **INSERTING A MENU OPTION**

To insert a menu option in a horizontal menu, move the cursor to the location where you want to place the new option and press [F1]. The highlighted option and all options to the right move right on a horizontal menu or down on a vertical menu. Enter the text for the new option.

### **CHANGING A MENU OPTION**

Move the cursor to the menu title you want to change and type in the new menu option text.

### **DELETING A MENU OPTION**

To delete a menu option, move the cursor to the option and press [F2].

When all changes are complete, press [ESC] to continue. After you finish modifying menu options, the EXPRESS displays the following message:

```
Do you want to change ESC or HELP action:  Yes  No
```

Select *Yes* if you want to change the escape method from a menu or if you want to build, change, or delete a help screen.

### **Changing the Escape Action**

You may want to change the function of the [ESC] key if an *Exit* option is added or deleted from a menu. The EXPRESS displays the message shown in the following screen. If [ESC] is to be used to exit the menu, answer *Yes*.

```
Is ESC to be used to exit this menu:  Yes  No
```

## Changing the Help Action

When you change the help action, the EXPRESS displays the following message:

```
Is HELP to be defined for this menu:  Yes  No
```

If you want to delete existing help text, answer *No*. If help text does not exist and you want to enter it, or if you want to change existing text, select *Yes*. If no help text exists, the EXPRESS prompts you to name the help text. Enter a one to eight character name. If possible, assign a name that is easily associated with the text. The name of each help screen in an application must be unique. After you enter the name, press [ENTER] and type the help text on the screen. You may enter 23 lines of text, and each line can contain as many as 78 characters. In the following screen, 14 lines of sample help text have been entered. Notice the cursor location information in the angle brackets in the upper-right corner. The first number indicates the line, and the second number refers to the column. When you are finished entering text, press [ESC].

```
Enter/edit the help text for this menu          < 14,  1 > [ESC] to exit
      Help for the Maintain Sales Representative Menu
Select ADD to enter data. Browse through the data to edit
any errors, and print a copy of the entered and edited data.

Select CHANGE to edit a record. Enter a valid
sales representative employee ID number.

Select REMOVE to remove a column. Enter the name
of a sales representative.

Select INQUIRY to display a record. Enter an
employee ID number.
```

## Changing an Existing Action

Next, the EXPRESS enables you to change the actions associated with each of the menu picks. As an example, suppose the *ADD* option of the menu shown on the screen below has one action, an EXPRESS *Load* action. In addition, you want the user to be able to browse through the entered data, edit any observed errors, and print the entered and edited data values.

You can accomplish this by adding two actions, the EXPRESS *Browse* and *Print* actions, after the currently defined *Load* action. As shown in the following screen, the EXPRESS displays the menu you are changing at the top of the screen and asks if you want to change any option actions. To install the two new actions needed for the *Load* action, select *Yes*.

Maintain Sales Representative Data			
ADD	CHANGE	REMOVE	INQUIRE

Do you want to change the pick actions: Yes No

The EXPRESS displays the following screen. Select *No* because you only want to change the pick actions for one menu option.

Do you want to review all the pick actions: Yes No

Next, the EXPRESS allows you to choose the menu selection that contains the action you want to change. In the following screen, move the cursor to *ADD* and press [ENTER].

Maintain Sales Representative Data			
ADD	CHANGE	REMOVE	INQUIRE

The EXPRESS displays the defined actions for the *ADD* option. *ADD* is highlighted in the menu at the top of the screen and the following screen is displayed. In this example, select *No* because you do not want to insert any actions before *Load*.

ACTION—Load using form: salesf1  
Do you want to insert actions before current action: Yes No

You are now able to retain, change, or delete the existing action. Select *Keep*. To change a custom or macro option, you must select *Keep*. Then, you have the option to edit the code.

ACTION—Load using form: salesf1

Select option for this action

Keep Replace Delete
---------------------

Next, you have the opportunity to install an action after the current action. Select *Yes* in order to install the *Browse* action. Then the EXPRESS lists the available actions. Select *Browse*.

The sequence of menus and prompts that enable you to define the *Browse* action are described in the “Defining a New Application” section. When you finish the definition, the EXPRESS asks if you want another action for the current menu option.

Select *Yes* to install the *Print* action. The actions are displayed once again. This time select and define *Print*. When you are finished, answer *No* when you are asked if you want to install another action. When you are prompted to change another menu in the application, choose *No* again. When the EXPRESS asks if you want to save the changes, press [ENTER]. The application files are updated and compiled. To exit without making changes, press [ESC].

## Removing a Menu from the Application

Select option 2 from the application change menu to remove a menu and all its submenus. The EXPRESS displays a list of all menu names in the application except the main menu. Move the cursor to the menu you want to delete and press [ENTER]. You are not given an opportunity to remove the main menu, because that would be the same as removing the entire application. To remove an application, use the ERASE command to erase the three application files, the *.app*



file, the *.apx* file, and the *.api* file. Forms and reports created for the application can be removed from the *forms* and *reports* tables in the R:base database by using the R:base DELETE ROWS command.

## DISPLAYING THE FILE DIRECTORY

Select option 5 to display the names of the files in the default directory. Below is a screen that shows a typical directory display:

```

C:COMPUC01.RBS  C:COMPUC02.RBS  C:COMPUC03.RBS  C:MAINMENU.TXT
C:MAINHELP.SCR  C:SALES.CMD      C:CUSTOMER.CMD   C:STARTUP1.CMD
C:MAINMENU.MNU  C:MAINHELP.SCR   C:STARTUP1.API    C:STARTUP1.APX
C:STARTUP1.APP  C:COMPUC01.PRC

```

Note the following filename extension conventions:

- ASCII application files produced by the EXPRESS have the extension *.app*.
- ASCII binary files produced by the EXPRESS have the extension *.apx*.
- ASCII binary files produced by the EXPRESS for internal use have the extension *.api*.
- R:base database files have the extension *.rbs*.

Filename extensions for other types of files are up to you. In the directory shown above, the following extension naming conventions are used:

- ASCII application files have the extension *.app*.
- ASCII text files used by R:base command files have the extension *.txt*.
- Command files have the extension *.cmd*.
- Procedure files produced by RCOMPILE have the extension *.prc*.
- R:base 5000 screen files have the extension *.scr*.
- R:base 5000 menu files have the extension *.mnu*.

## EXITING THE EXPRESS

Select option 6 to exit the EXPRESS. If you entered the EXPRESS from the R:base 5000 main menu, selecting option 6 returns you to the same menu. If you entered the EXPRESS command at a DOS prompt, selecting option 6 returns you to an operating system prompt.

# The R:base 5000 Programming Language Contents

<b>How to Use This Chapter</b>	15-3
<b>Command Files</b>	15-3
<b>RBEDIT</b>	15-4
Running a Command File	15-7
Modifying a Command File	15-7
<b>Comments</b>	15-8
<b>Variables</b>	15-8
Variable Names	15-8
Creating Variables	15-9
Variable Data Types	15-9
Explicit Data Typing	15-9
Implicit Data Typing	15-10
Changing a Variable's Data Type	15-10
Holding Values	15-11
Column Values in Tables	15-11
Results of Expressions	15-11
Display Text	15-11
Constructed Values	15-12
System Variables	15-13
Keyboard Input Values	15-13
<b>Control Structures</b>	15-13
Conditional Command Execution	15-13
IF . . . ENDIF	15-13
Rules for Constructing Conditions	15-14
IF . . . ELSE . . . ENDIF	15-15
WHILE . . . ENDWHILE	15-17
DO CASE	15-18
GOTO Command	15-20
Nesting Considerations	15-20
<b>Input/Output</b>	15-22
One Keystroke Acknowledgment	15-22
FILLIN Command	15-23
Menus	15-23
The WRITE and FILLIN Commands	15-24
The TYPE and FILLIN Commands	15-24
The DISPLAY and FILLIN Commands	15-25
The CHOOSE Command	15-25

<b>Changing R:base Prompts</b>	15-27
Changing the PROMPT.DAT File	15-28
Structure of a Prompt Block	15-28
Prompt Blocks with Parameters	15-29
Prompt Blocks That Display Other Prompt Blocks	15-31
Generating a Prompt File	15-32
<b>Variable Forms</b>	15-32
The FORMS Command	15-32
The DRAW Command	15-35
The ENTER VAR Command	15-36
The EDIT VAR Command	15-37
Command File Examples	15-38
From Paper Form to Multiple Database Tables	15-38
Data Integrity and Security	15-40
Table Lookup and Doing Calculations	15-42
Multiple Page Form Emulation	15-44
<b>Database Access</b>	15-45
<b>Messages and Errors</b>	15-46
<b>Debugging</b>	15-48
<b>Application Integration</b>	15-48
Subroutines	15-48
Macros	15-50
The Initial Command File	15-51

## HOW TO USE THIS CHAPTER

This chapter, a user's guide to the R:base 5000 programming language, provides information on how to build and run command files, on variables, variable forms, control structures, output and input, R:base prompts, database access, messages and errors, debugging, and topics on managing applications. The section provides numerous samples of R:base command files, and it also includes information on how to use RBEDIT, the R:base editor.

The *R:base 5000 Reference Manual* "Command Dictionary" supplements the information in this chapter. For example, several command file segments are shown in this chapter which demonstrate fundamental R:base 5000 programming techniques. To understand the segment so you can apply it to your own work, use the "Command Dictionary" for information about individual commands.

## COMMAND FILES

R:base can execute commands one at a time as they are entered from the keyboard and can also execute a list of commands stored in a file on a disk. This file is called a command file.

When you enter an R:base command from the keyboard, R:base executes the command immediately. For example, if at the R> prompt you enter

```
NEWPAGE
```

the screen is immediately erased and the R> prompt for the next command appears in the upper left-hand corner of the screen.

This is satisfactory when a single command will do the job. Some tasks, however, require more than one command or are performed more than once. For example, you can accomplish these tasks

- Clear the screen
- Open a database
- Display the contents of one of the database tables

with these R:base commands:

```
NEWPAGE
OPEN compuco
SELECT ALL FROM salesrep
```

If you do this from the keyboard, you have to enter the first command, wait until it is finished executing, enter the second command, wait for it to finish, and then enter the third command. After entering the third command, the contents of the database table appear on the screen:

empid	frstname	lastname	address	city
102	June	Wilson	3278 Summit Drive	Seattle
129	Ernest	Hernandez	12390 Windermere Dr.	Seattle
131	John	Smith	3050 N.E. 41st	Seattle
133	Peter	Coffin	4105 29th Ave N.E.	Bellevue
160	Mary	Simpson	101 West Mercer	Redmond

You can accomplish the same task by saving the commands in a command file and using the R:base RUN command to execute the command file. For example, if the command file *opn.cmd* contains the NEWPAGE, OPEN, and SELECT commands, then you can execute those commands by entering:

```
RUN opn.cmd
```

R:base will read the disk file *opn.cmd* and execute the commands just as if you had entered them from the keyboard.

## RBEDIT

RBEDIT, the R:base editor program, is designed to build command files. Other editors can also be used to build R:base command files. If you use another editor, make sure that the command file contains only ASCII characters, and no embedded editor commands or control characters.

To use RBEDIT, you can enter *rbedit* at the DOS prompt or at the R > prompt. From the R:base main menu, select RBEDIT.

The RBEDIT menu looks like this:

```
---Old File---New file---Quit-----R:base screen editor
```

To build a new command file, move the cursor to the *New file* option and press [ENTER]. Your screen is blank, except for the message in the upper right-hand corner:

```
<1, 1> [ESC] to exit
```

The two numbers inside the angle brackets indicate the position of the cursor in the file. The first number is the line number. The second number is the column number (1 through 80).

RBEDIT can create or modify a command file of up to 800 command lines, depending on available memory. The screen displays 23 lines at a time.

The RBEDIT key functions are shown in table 15-1.

Table 15-1 RBEDIT Key Functions

To browse through pages:\*

[Home]	Display the first page of the text file
[End]	Display the last page of the text file that is shown by moving to the last line in the file
[PgUp]	Display the previous page by moving up 20 lines in the file and displaying 23 lines
[PgDn]	Display the next page by moving down 20 lines in the file and displaying 23 lines

To move the cursor in line units:

[↓]	Move down one line
[↑]	Move up one line
[Ctrl] [→]	Move to end of current line
[Ctrl] [←]	Move to start of current line
[ENTER]	Move to start of next line

To move the cursor in character units:

[→]	Move right one character
[←]	Move left one character
[↘]	Move right to start of next column (columns start at multiples of 10)
[↙] [↔]	Move left to start of previous column

To edit a line:

[F1]	Insert a line above the current line
[F2]	Delete the current line

To edit a character:

[Ins]	Insert a blank character
[Del]	Delete a character
[F4]	Character repeat mode toggle switch; when this switch is on, you can repeat a character by pressing the arrow keys

\*RBEDIT displays 23 lines at a time on the screen.

If you enter these command lines

```
NEWPAGE
OPEN compuco
SELECT ALL FROM salesrep
```

Your screen will appear as:

```

NEWPAGE
OPEN compuco
SELECT ALL FROM salesrep
                                     <4, 1> [ESC] to exit
```

Now press [ESC].

RBEDIT displays this menu:

```

-----Edit again-----Save file-----Next file-----Quit-----R:base screen editor
```

- Select *Edit Again* if you want to change the file you were just editing
- Select *Save File* to save the file you were just editing on disk
- Select *Next File* to get prompted for the name of a new file to edit
- Select *Quit* to return to either the R> prompt or the R:base main menu

Move the cursor to the *Save file* option and press [ENTER].

RBEDIT prompts you for a file name. (The filename can be preceded by a drive designator and a path.) For example, enter *exp.cmd* to save the command file under that file name in the current directory on the default drive.

```

-----Edit again-----Save file-----Next file-----Quit-----R:base screen editor
Name of new file to save:exp.cmd
```

After saving the file, RBEDIT displays the following menu:

```

-----Old file-----New File-----Quit-----R:base screen editor
```

Move the cursor to the *Quit* option and press [ENTER].

## Running a Command File

To execute an existing command file, at the R> prompt enter the RUN command. For example, to execute the three-line command file *exp.cmd*, enter:

```
RUN exp.cmd
```

The display that appears on the screen is identical to the one that resulted from entering the three commands one at a time from the keyboard:

Database exists				
empid	frstname	lastname	address	city
102	June	Wilson	3278 Summit Drive	Seattle
129	Ernest	Hernandez	12390 Windermere Dr.	Seattle
131	John	Smith	3050 N.E. 41st	Seattle
133	Peter	Coffin	4105 29th Ave N.E.	Bellevue
160	Mary	Simpson	101 West Mercer	Redmond
Switching INPUT back to KEYBOARD				

R:base executed each command just as it would had you typed them at the keyboard. The message *Switching INPUT back to KEYBOARD* indicates that R:base has finished executing the command file and is now waiting for more commands to be entered.

## Modifying a Command File

Often, you need to change the content of a command file you have saved on disk. Command files evolve as the people using them get new ideas and as the database management environment changes.

For example, to edit an old command file named *exp.cmd*, call up RBEDIT.

From the RBEDIT menu, select *Old file*. RBEDIT prompts you for the name of the old file to edit. Enter the filename *exp.cmd* at the prompt and press [ENTER].

The screen briefly displays the message *Reading File*. Then, the command file text appears on the screen. You can now change your command file text.

To bring an existing file to the screen, you have the option of typing *RBEDIT* and the name of the command file at the R> prompt. For example:

```
RBEDIT exp.cmd
```



To change the command file text, use the RBEDIT key functions listed in table 15-1. When you finish editing, press [ESC]. RBEDIT displays a menu which gives you the opportunity to save the changed version of the file on disk under its old file name or to create a new file.

## COMMENTS

Command files may contain comments. An R:base command file comment starts with the characters `*(` and ends with a `)`. A comment is not executed when the command file is run. For more information, see the *R:base 5000 Reference Manual* "Command Dictionary."

## VARIABLES

Variables are stored in memory in a variable table that holds data separate from the values in a database. Because variables are stored in memory, they have certain characteristics that differentiate them from data stored in a database.

- They are lost when you exit R:base.
- They can be used by many other functional processes in R:base. Thus, they are often referred to as global variables.
- They can be cleared with the CLEAR command.

## Variable Names

R:base commands use variables in two different formats: the variable name, *varname*, and the value the variable contains, *.varname*, a dotted variable.

The following commands use the variable name as part of the syntax of the command:

```
SET VAR varname...
SHOW VAR varname
IF varname...THEN
WHILE varname...THEN
SET ERROR VAR varname
SET POINTER #n varname FOR tblname...
FILLIN varname...
DRAW variable formname WITH varname1 varname2...
COMPUTE varname...
CHOOSE varname...
EDIT VAR varname1 varname2...
ENTER VAR varname1 varname2...
SHOW ERROR varname AT...
```

When you use a variable in any other R:base command, precede the variable with a dot. A dotted variable causes the current value of the variable to be used. For example, if *v1* = *jobs* and *v2* = *sales*, then the execution of the commands

```
OPEN .v1
SELECT ALL FROM employee WHERE dept EQ .v2
```

opens the database *jobs* and selects rows from the *employee* table where *dept* equals *sales*.

## Creating Variables

The following commands create a variable:

```
SET VARIABLE...
FILLIN varname...
COMPUTE varname...
CHOOSE varname...
SET ERROR VARIABLE...
SET POINTER #n varname...
```

The following commands create one or more variables:

```
DRAW...
EDIT VAR...
ENTER VAR...
```

The number of variables that can be defined at any one time is dynamic. The factors determining the number are available memory and the length of the variable value.

## Variable Data Types

Variables have data types, just like database columns. You can establish a data type explicitly or implicitly. You can also change the data type of an existing variable.

### EXPLICIT DATA TYPING

The explicit method of variable data typing is suggested, because this prevents ambiguity. Variable data types are set explicitly with the command of the general form *SET VAR varname datatype*. For example:

```
SET VAR v1 TEXT
```

This command defines a variable as TEXT and sets its value to null. Note that this is only true if the variable does not already exist or if changing the data type of an existing variable to a new data type creates invalid values.

The command is often followed with a command that sets a value. For example:

```
SET VAR v1 TEXT ; SET VAR v1 TO SEATTLE
```

The keyword TO differentiates between the two uses of the SET VAR commands.

### **IMPLICIT DATA TYPING**

Both these functions, setting a data type and setting a value, can be done with the SET VAR TO command form alone. R:base sets the data type to the type implied by the value. For example, the following command causes *v1* to be typed as REAL because the initial value, *1.0*, contains only numbers and a decimal point.

```
SET VAR v1 TO 1.0
```

The rules R:base uses to assign data types to new variables based on their value are as follows:

1. An INTEGER data type results when the value contains only numerals (0, 1, 2, ...,9) and there are nine or fewer numerals in the number.
2. A DOLLAR data type results when:
  - The first character of the value is a dollar sign (\$) followed by numerals
  - A numeric value with more than nine and less than 17 numerals
3. A REAL data type results when:
  - A numeric value contains a decimal point (.) followed by numerals or blanks
  - A numeric value begins with a decimal point (.)
  - The value is a decimal point (.), interpreted as 0.0
4. A TIME data type results when the value is a valid time range with : as delimiter.
5. A DATE data type results when the value is a valid date range with \*, /, -, or no spaces as delimiters.
6. A TEXT data type results when the value cannot be assigned another data type using the rules above.

### **CHANGING A VARIABLE'S DATA TYPE**

You can change a data type you have explicitly defined or one that R:base has implicitly defined by using the following form of the SET command.

```
SET VAR varname datatype
```

When you change a data type for a variable, R:base stores the value as the new type if it can. If the value is incompatible with the new type, R:base will store a null value for the variable. For example:

- Changing the TEXT value *"This is a string"* to any other data type results in a null value.
- Changing the TEXT value *5555.11* to INTEGER results in the value *5555*, to DOLLAR results in *\$5,555.11*, and to REAL in the value *5555.11*. Changing it to DATE or to TIME, however, results in a null value.
- Changing the date value *12/15/85* to INTEGER results in a null value. Changing it to TEXT, however, results in the value *12/15/85*.

## Holding Values

Following are some programming examples that illustrate each of the ways variables are used.

### COLUMN VALUES IN TABLES

To move a column value to a variable, use one of the following general forms of the SET VAR command.

```
SET VAR varname TO colname IN tblname WHERE condlist
SET VAR varname TO colname IN #n
COMPUTE VAR varname AS MAX colname FROM tblname
```

### RESULTS OF EXPRESSIONS

An expression combines two items using an arithmetic operator (+, −, ×, /, or %). The items have numeric data types (INTEGER, REAL, DOLLAR, and in some cases DATE or TIME). They may be values or dotted variables. For example:

```
SET VAR v TO 11
SET VAR v TO .v1 + 1
SET VAR v TO .v1 + .v2
```

The data type of the result depends upon the data types of the items in the expression. See chapter 9 and the SET command in the *R:base 5000 Reference Manual*.

### DISPLAY TEXT

In the following example, a command file segment defines a TEXT variable to hold a display string, such as the title or heading, and displays it at a specific location on the screen.

```
SET VAR v1 TEXT
SET VAR v1 TO "table of geographic locations"
SHOW VAR v1 AT 12,4
```

**CONSTRUCTED VALUES**

Variables with data type TEXT can hold characters which are a combination of characters from two other variables. Use the string concatenation operators + and & in a SET VAR command to do this. For example, suppose the column *frstname* in table *clients* contains the first name of a client and the column *lastname* contains the last name. The following command file segment moves the column values into variables and then combines the first and last names in another variable. This would be useful in a mailing list program, for example. The & operator is used to put a space between the first name and the last name.

```
SET VAR vfirst TO frstname IN clients WHERE clientno EQ 675
SET VAR vlast TO lastname IN clients WHERE clientno EQ 675
SET VAR vfullnam TO .vfirst & .vlast
```

The following segment of commands combine the *city*, *state*, and *zip* columns from a table into one variable, along with a comma between the city and state and two spaces between the state and zip. The following command segment assumes that the SET POINTER command has been used to locate the table row. For more information about the SET POINTER command, see the *R:base 5000 Reference Manual* "Command Dictionary."

```
SET VAR vcity TO city IN #3
SET VAR vstate TO state IN #3
SET VAR vzip TO zip IN #3
SET VAR vcity TO .vcity + " , "
SET VAR vzip TO " " & .vzip
SET VAR vcity TO .vcity & .vstate
SET VAR vcity TO .vcity & .vzip
```

You can use the MOVE command to extract substrings from variables with type TEXT. Suppose the column *partno* in the table *partlist* has a ten-character format and the third, fourth, and fifth characters are a supplier code. To move these three digits into a variable, the following segment of a command file can be used:

```
SET VAR vpartno TO partno IN #2
MOVE 3 FROM vpartno AT 3 TO vsupcode AT 1
```

In this case, the MOVE command moves three characters, starting with the third character in *vpartno* to the first three characters of *vsupcode*.

Another application of the MOVE command is to build a display line in a variable. Then the entire line can be displayed on either a screen or printer without a carriage-return/line-feed break between items in the line.

## SYSTEM VARIABLES

The system variables #TIME and #DATE contain the system time and date. They can be used like any other variable. They should not be assigned a value because they are set and updated by the system hardware.

## KEYBOARD INPUT VALUES

See the descriptions of the CHOOSE, FILLIN, EDIT VAR, and ENTER VAR commands in the "INPUT/OUTPUT" section of this chapter.

## CONTROL STRUCTURES

There are four general types of control structures in the R:base programming language:

- Conditional execution (IF...THEN...ENDIF)
- Conditional repetitive execution (WHILE...THEN...ENDWHILE)
- Unconditional transfer of control (GOTO *labelname* and LABEL *labelname*)
- Subroutine calling and return (RUN...RETURN)

## Conditional Command Execution

### IF...ENDIF

The form of the IF...ENDIF block is:

```
IF condlist THEN
  then-block
ENDIF
```



*Condlist* is a logical statement (either true or false) about one or more variables. *Then-block* is a list of commands that are executed if and only if the logical statement is true. If the logical statement is false, the commands between THEN and ENDIF are not executed and execution resumes with the command immediately after the ENDIF.

The IF...ENDIF structure may be on a single line in the command file. For example,

```
IF x = 0 THEN ; y = y + 1 ; ENDIF
```

The only exception to this is if the last command in the *then-block* is QUIT. If so, you must place the ENDIF on a separate line.

Below is an example command file segment:

```
WRITE "Preparing report. Please wait. . ." AT 5 1
IF device EQ "PRINTER" THEN
  OUTPUT PRINTER
ENDIF
SET VAR valid TO "NO"
```

Figure 15-1 shows a flowchart that shows the flow of control through the above command file segment.

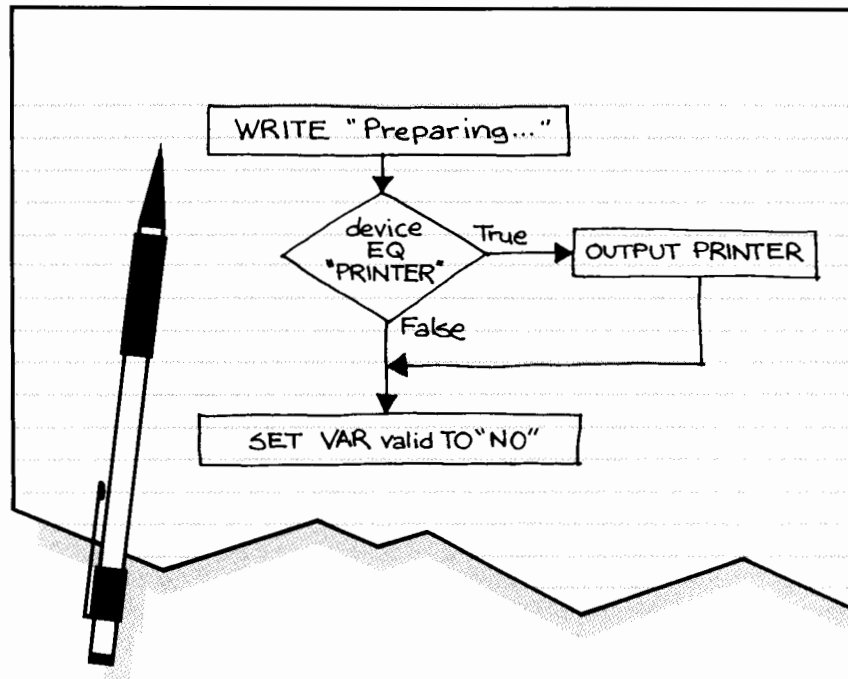


Figure 15-1 IF...ENDIF Sample Flow Chart

#### **RULES FOR CONSTRUCTING CONDITIONS**

Table 15-2 lists the conditions that can be used in an IF...THEN...ENDIF control structure.

Table 15-2 Kinds of Conditions That Can Be Used in IF Commands

Kind of condition	Condition is True if . . .
varname EXISTS	The value of the variable is other than null.
varname FAILS	The value of the variable is null.
varname CONTAINS string	The variable is of data type TEXT, and <i>string</i> is contained as a substring within the text string currently stored in the variable.
varname <i>op</i> value <i>op</i> is EQ, NE, GT, GE, LT, LE, =, >, <, >=, or <=	The value of the variable has the specified relationship (equal, not equal, etc.) to a value.
varname1 <i>op</i> varname2 <i>op</i> is EQ, NE, GT, GE, LT, LE, =, >, <, >=, or <=	The values of the two variables have the specified relationship (equal, not equal, etc.)

Up to ten conditions may be combined in a single IF command by using AND and OR. An example of simple conditions combined with AND and OR logical operators into one *condition* is shown below.

```
IF firstname EQ Mary OR firstname EQ John AND lastname EQ Smith THEN
.
.
.
ENDIF
```

The *condition* is true only when the value of the variable *firstname* is Mary or John and the value of the variable *lastname* is Smith.

Parentheses cannot be used with multiple conditions.

### IF...ELSE...ENDIF

The IF...ELSE...ENDIF control structure controls the flow through to mutually exclusive command segments. This implements the following case:

IF true, do this; otherwise, do that.

The general form of such a structure is:

```
IF condlist THEN
    then-block
ELSE
    else-block
ENDIF
```

*Condlist* is a logical statement. The possible conditions are summarized in table 15-2.



*Then-block* is a list of commands that are executed if and only if the logical statement is true. If *condlist* is false, *else-block* is executed. Either the *then-block* or the *else-block* is executed. Then processing continues with the command line immediately following the ENDIF.

Here is an example:

```
SET VAR shiprate TO shiprate IN #3
IF shiprate FAILS THEN
  SET VAR freight DOLLAR
  SET VAR freight TO 0
ELSE
  SET VAR freight TO .shiprate x .ordamt
ENDIF
WRITE "-----" AT .rownum 64
```

Figure 15-2 is a flowchart that shows the flow of control through the command segment above.

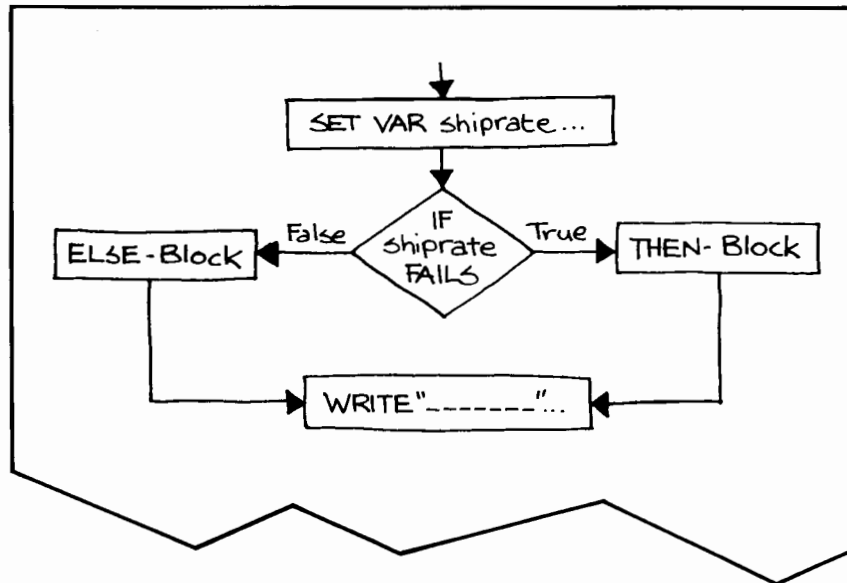


Figure 15-2 IF...THEN...ENDIF Sample Flow Chart

#### **WHILE...ENDWHILE**

The WHILE...ENDWHILE structure is used to execute commands repetitively as long as a given condition is true.

The general form of the WHILE...ENDWHILE structure is:

```
WHILE condlist THEN
  while-block
ENDWHILE
```

While *condlist* is true, R:base repeatedly executes the *while-block*. Each time the commands are executed, the WHILE...THEN statement is tested. If *condlist* is true, the commands between THEN and ENDWHILE are executed again.

You can use a WHILE...ENDWHILE to repeat a list of commands as long as the condition is true. This is called looping. Examples are shown below.

This example executes ten times:

```
SET VAR count TO 0
WHILE count < 10 THEN
  SET VAR count to .count + 1
  .
  .
ENDWHILE
*(next command)
```

Figure 15-3 is a flowchart that shows the flow of control through the command segment.

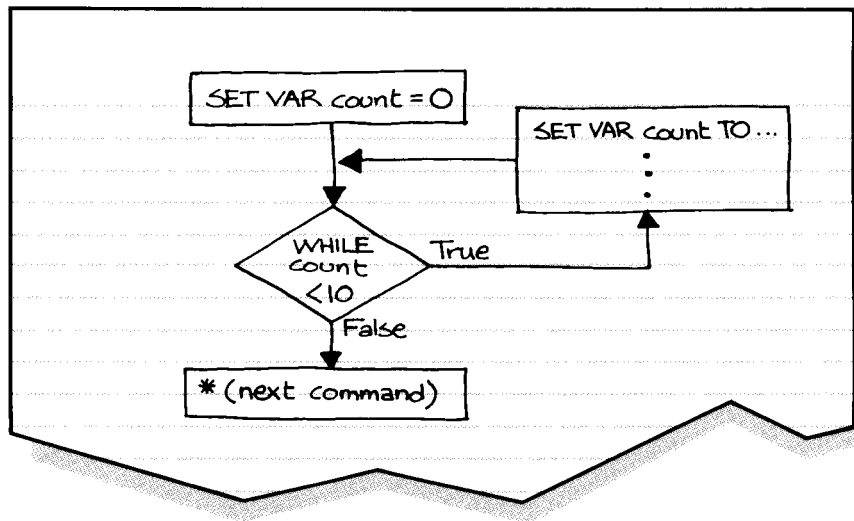


Figure 15-3 WHILE...ENDW Sample Flow Chart

In this example, the loop continues until *s* is set to a non-zero value. The **BREAK** command breaks out of the while loop it is in.

```
SET VAR s TO 0
SET VAR t TO more
WHILE t EQ more THEN
    .
    .
    .
    IF s NE 0 THEN
        BREAK
    ENDIF
    .
    .
ENDWHILE
```

It is not a good programming practice to use a **RETURN** command within a **WHILE** loop or an **IF...THEN** structure. The **WHILE** loop will not be cleared and, therefore, may cause problems during command file execution. Use a **GOTO** or **BREAK** instead.

### **DO CASE**

You can use a table in an R:base database together with some simple commands to do the equivalent of a **CASE** statement found in some other programming languages. Typically the form of a case is that for each value or case some action is done. Here is an example:

```
Case 1
  do action for case=1
Case 2
  do action for case=2
Case 3
  do action for case=3
etc.
```

There is no problem with the logic of a **DO CASE** construction, but from a code implementation standpoint, you end up writing many lines of code. Any time you add another case, you have to add code. Here is an alternative method that can be used in R:base which duplicates the logic of a **DO CASE** but eliminates many lines of code, giving you an open-ended structure for easily adding cases.

First, you define a table containing the value of the case and the input file to execute for that case. For example:

```

DEFINE
COLUMNS
case INTEGER
file TEXT 15
TABLES
cases WITH case file
END
LOAD cases
1 file1.cmd
2 file2.cmd
3 file3.cmd
.
.
.
END

```

With such a table as part of your database, the DO CASE code shrinks down to this:

```

SET VAR choice INTEGER
SET VAR infile TO keyboard
FILLIN choice USING "What case do you want?"
SET VAR infile TO file IN cases WHERE case EQ .choice
RUN .infile

```

These five lines of code can handle all the cases you might need. Having another case simply means adding an entry to the *cases* table. You do not need to add code to the DO CASE code.

The variable *choice* is preset to INTEGER because only INTEGER is to be accepted by the FILLIN command. The variable *infile* is preset to *keyboard* because, if the user chooses a case which does not exist, the SET VAR *infile*... command will leave the value of *infile* unchanged and control will revert back to the keyboard, rather than try to execute a non-existent case. You may want a range of values to check and give an error message if an invalid case is entered.

## GOTO Command

The GOTO and LABEL commands also can be used to transfer control. A GOTO command causes processing control to be passed to the command line immediately following the corresponding LABEL *labelname* command. The LABEL command may precede or follow the GOTO command in the same command file only.

The GOTO command executes more slowly than other R:base control structures, and each GOTO command should be executed only once in a file and not be used for looping.

The general form of the GOTO command is:

```
GOTO labelname
```

```
.  
.  
.
```

```
LABEL labelname
```

LABEL identifies a command line to which control is passed from a GOTO statement. *Labelname* is a one-to-eight character string.

## Nesting Considerations

Nesting means locating a control structure within a similar control structure—for example, an IF block within another IF block. Command files, IF blocks, and WHILE blocks can be nested. For all three kinds of nesting there is a maximum of ten nesting levels. The first level is zero (0) and the last level is nine. For command files there is a maximum of six nesting levels (zero through five).

When a control structure or command file is added to the nesting, its nesting level is increased by one. The commands that cause a nesting level increase are WHILE, IF, INPUT, QUIT TO, and RUN.

When a control structure or command file is removed from the nesting, its nesting level is decreased by one. The commands that decrease a level are ENDWHILE, BREAK, ENDIF, and RETURN. When a COMMAND file terminates—runs out of commands to execute—it does an automatic RETURN back to the COMMAND file that called it with an INPUT or RUN command.

QUIT and GOTO also affect these nesting levels. QUIT clears all nestings. GOTO reads the command file it is in, looking for its matching LABEL. As it reads through the file, it decreases one level for each ENDWHILE and ENDIF it encounters and increases one level for each WHILE and IF it encounters. Table 15-3 lists the control structures.

Table 15-3 Control Structure Table

Command	WHILE	IF	COMMAND-FILE
ENDW	-1	....	....
BREAK	-1	....	....
ENDIF	....	-1	....
QUIT	-all	-all	-all
QUIT TO	-all	-all	-all +1
RETURN	....	....	-1
EOF	....	....	-1
WHILE	+1	....	....
IF	....	+1	....
INPUT	....	....	+1
RUN	....	....	+1

-1 means to decrease one nesting level

+1 means to increase one nesting level

-all means to clear all nesting levels

Also, see GOTO.

Examples of nested structures are illustrated in the sections below. Other good examples that show nesting and complex control structures can be found in the samples files provided on one of the R:base 5000 disks. See the ORDER application files and the *convert.cmd* program file.

If a command file is interrupted by an error or by pressing the escape key, [ESC], repeatedly, the command processor may still be inside an IF, WHILE, or INPUT structure, or in the LOAD mode. When this occurs, the system will appear to be stuck and may produce confusing error messages. You may get out of these situations by entering one of the following commands at the R> prompt:

- QUIT will release from INPUT
- ENDIF will exit from one level of IF
- END will exit from LOAD
- ENDWHILE will exit from one level of WHILE

An I> or W> prompt indicates that an IF structure (I>) or a WHILE structure (W>) was interrupted. Entering ENDIF or ENDWHILE clears the appropriate nesting structure stack to yield an R>.

To prevent the interruption of a critical process or to prevent an undesired hard exit from a command file, you can disable [ESC] with the SET ESCAPE OFF command. This does not affect the [ESC] function used in menus. The [ESC] key function is restored with SET ESCAPE ON.

## INPUT/OUTPUT

R:base has several commands that output a screen display or display and pause until the operator enters some input. These commands are:

- WRITE       —Displays a message
- PAUSE       —Suspends execution of the command file until the operator presses a key
- TYPE        —Displays the contents of an ASCII file
- DISPLAY     —Displays a screen file or block
- FILLIN      —Prompts for a variable value and accepts operator input
- CHOOSE     —Displays a menu and accepts operator input
- ENTER       —Inputs column values from a table form
- EDIT        —Inputs changes to columns from a table form
- DRAW        —Displays a variable form and variable values
- ENTER VAR   —Inputs values to a variable form
- EDIT VAR    —Inputs changes to variables to a variable form

The kinds of output produced include: prompt line, screen, menu, table form, and variable form.

The kind of response expected from the operator includes:

- One keystroke to acknowledge the output
- One keystroke to make a menu choice
- Characters into one area of the screen
- Characters into several areas of the screen

### One Keystroke Acknowledgment

It is important to give the operator time to read messages and screens and to acknowledge that they are understood before the command file continues executing. You can use a PAUSE command to get the command file to suspend execution until the operator presses a key.

A WRITE command can be used to prompt the operator to press a key when the operator wants the command file to continue. For example:

```
NEWPAGE
DISPLAY helpfile
WRITE "Press any key to continue" AT 24 10
PAUSE
NEWPAGE
```

The message that the operator acknowledges having received does not have to be a simple output by one WRITE command. It can be an entire screen which will help complete a task. This information does not have to be in the command file itself. It can be stored:

- In a separate file
- In a block separate from the command block in an application file built by the EXPRESS
- In a block separate from the command block in a procedure file output by RCOMPILE

A TYPE command can be used to display text only if it is in a separate file. A DISPLAY command can be used to display text from all three.

## FILLIN Command

The FILLIN command enables the command file to record the operator's response and use it later. The general form of the FILLIN command is:

```
FILLIN varname USING "prompt message" [ AT scrnrow scrncol ]
```

The variable *varname* records the operator's response to the prompt message. The prompt message begins at the specified screen row and column position.

The variable is set to the data type that matches the operator entry. If you enter 123, for example, the variable is typed as INTEGER. If you want the data type as TEXT in this case, you must surround the numeric response with quotation marks. Entering "123" would set the data type to TEXT.

Below is an example of a command file segment which uses a FILLIN command to display a prompt line and record a response. The FILLIN command is used in place of the WRITE and PAUSE commands:

```
NEWPAGE
FILLIN var1 "Enter the name of the database to open" AT 24 10
OPEN .var1
QUIT
```

## Menus

There are several ways in R:base to implement menus:

- Use WRITE commands and a FILLIN command
- TYPE an ASCII file and use a FILLIN command
- DISPLAY a text block and use the FILLIN command
- Use a CHOOSE command and a menu system

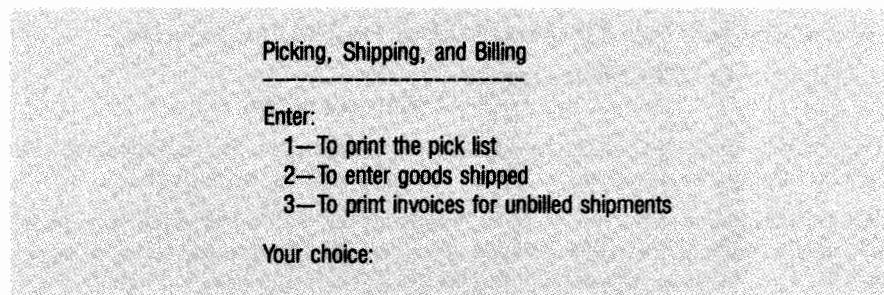


**THE WRITE AND FILLIN COMMANDS**

You can combine the WRITE and FILLIN commands to create a menu as shown in the command file segment below.

```
WRITE "Picking, Shipping, and Billing" AT 2 20
WRITE "-----" AT 3 20
WRITE "Enter:" AT 5 20
WRITE " 1—To print the pick list" AT 6 20
WRITE " 2—To enter goods shipped" AT 7 20
WRITE " 3—To print invoices for unbilled shipments" AT 8 20
WRITE "Your choice: " AT 10 20
FILLIN choice USING " " AT 10 33
```

When executed, these commands will produce a menu that looks like the screen below.



```
Picking, Shipping, and Billing
-----
Enter:
 1—To print the pick list
 2—To enter goods shipped
 3—To print invoices for unbilled shipments
Your choice:
```

**THE TYPE AND FILLIN COMMANDS**

A menu like that shown above can be displayed and the operator's response recorded with the following command segment:

```
TYPE a:menu1.txt
FILLIN choice USING " " AT 10 33
```

This method uses fewer commands in the command file than the method which uses one WRITE command to display each line of the menu. It is also more flexible, because the menu displayed by the command file can be changed by changing the file named in the TYPE command.

**THE DISPLAY AND FILLIN COMMANDS**

The same menu can be displayed using the DISPLAY command. The text of this screen block that has been used as an example is shown below.

```

SCREEN1
      Picking, Shipping, and Billing
      -----
      Enter:
      1—To print the pick list
      2—To enter goods shipped
      3—To print invoices for unbilled shipments
      Your choice:

```

The first line of the screen block is the menu name. While the screen functions as a menu display, it is not a menu block. A screen block can only be displayed.

Below is the command segment for the menu function using a screen file.

```

DISPLAY a:screen1.scr
FILLIN choice USING " " AT 10 33

```

If you use a screen block produced by RCOMPILE, the commands used are:

```

DISPLAY screen1 IN a:proc1.prc
FILLIN choice USING " " AT 10 33

```

For more information about procedure files and screen blocks, see chapter 16.

**THE CHOOSE COMMAND**

The CHOOSE command displays menus of two types: vertical and horizontal. The vertical menu has the items listed vertically. The menu can be displayed and the operator's response recorded with one command. For example, the following command displays the menu stored in the ASCII file *b:menu1.mnu* and records the operator's selection in the variable *choice*.

```

CHOOSE choice FROM b:menu1.mnu

```

If the menu is stored as a menu block in a compiled menu or procedure file, an example of the CHOOSE command is:

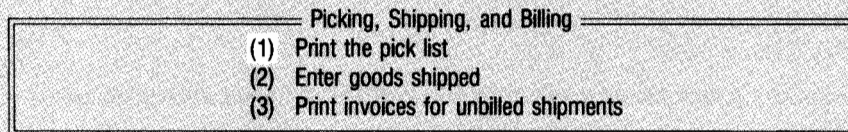
```
CHOOSE choice FROM menu1 IN b:proc1.prc
```

*Menu1* is the name of the menu block, and *b:proc1.prc* is the name of a compiled menu or procedure file that contains the block.

The text for a vertical menu file is shown below.

```
MENU1
COLUMN Picking, Shipping, and Billing
Print the pick list
Enter goods shipped
Print invoices for unbilled shipments
```

When the CHOOSE command is executed and the vertical menu is displayed, it looks like this:



```

Picking, Shipping, and Billing
(1) Print the pick list
(2) Enter goods shipped
(3) Print invoices for unbilled shipments
```

There are differences between the menu text and what appears on the menu display. The selection numbers (1), (2), and (3) are not in the menu file. The CHOOSE command generates them.

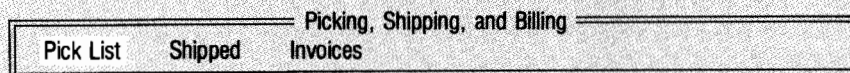
Also, there is no place for the operator to type in a menu selection. The selection is made by moving the reverse video bar until the selection is highlighted, and then pressing [ENTER]. Such vertical menus are described in chapter 14.

If the operator presses [ESC] when a menu is displayed by the CHOOSE command, the variable *choice* is set to 0. The command file can take appropriate action. If the operator presses [F10], *choice* is set to -1. The command file can detect that and may display a help screen. Otherwise, *choice* takes on a value of 1, 2, or 3, which the command file can process as the operator's selection. There can be up to nine selections in a vertical menu.

The other type of menu that may be displayed by the CHOOSE command is a horizontal menu. A menu file or block for a horizontal version of the example menu is shown below.

```
MENU2
ROW  Picking, Shipping, and Billing
Pick List
Shipped
Invoices
```

When the CHOOSE command is executed and the horizontal menu is displayed, it looks like this:



The image shows a graphical representation of a horizontal menu. It consists of a rectangular box with a title bar at the top that reads "Picking, Shipping, and Billing". Below the title bar, there are three buttons or options: "Pick List", "Shipped", and "Invoices". The "Shipped" button appears to be the currently selected option.

The selection process and the function of the [ESC] and [F10] keys are the same for a horizontal menu as for a vertical. Notice, however, that the menu text has been changed radically in the horizontal menu. That is because there can be no more than ten characters in the text for an option. An explanation of the short messages could be made in a help screen.

For a horizontal menu, *choice* is a TEXT variable and the operator's selections are returned as literal values. *Pick List*, *Shipped*, and *Invoices* in the example. If the operator presses [ESC], the value *ESC* will be put into *choice*. If the operator presses [F10], the value *HELP* will be put in *choice*.

## CHANGING R:BASE PROMPTS

To change R:base prompts, follow these steps:

1. Copy your current *prompt.dat* and *prompt.rbs* files and keep the copies in a safe place.
2. Use RBEDIT, or other editor, to change one or more prompt blocks in the *prompt.dat* file.
3. Run *promgen.exe* to convert the new *prompt.dat* file to a *prompt.rbs* file that can be displayed.
4. Run R:base and test the new prompts by entering PROMPT commands that refer to the new prompt blocks by the name of the block.

## Changing the PROMPT.DAT File

The easiest way to start working with the *prompt.dat* file is to change one of the prompt blocks in the file with RBEDIT. You may also add a new prompt block to the file. There may be up to 102 prompt blocks in the file.

### STRUCTURE OF A PROMPT BLOCK

The *prompt.dat* file is made up of prompt blocks. A prompt block has the following structure:

- The prompt block name is on line 1, starting in column 1.
- The command line to execute when the operator selects *Go* from the Prompt mode menu is on line 2, starting in column 1. There may be multiple commands on the command line, separated by semicolons. This 80-character line is all that can be used for commands; there can be no continuation lines.
- The actual prompt display (the text containing the message, a screen display, comments, or requests for input) starts on line 3 and uses up to 23 lines. All lines in this section must either be blank or begin in column two and finish before column 76.
- The last line of the block must contain the characters ENDC and must begin in column 1.

For example, below is the text for a prompt block:

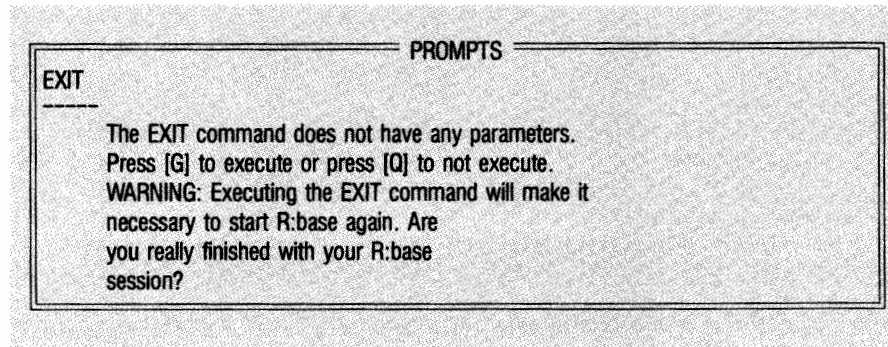
```
EXIT
EXIT
    The EXIT command does not have any parameters.
    Press [G] to execute or press [Q] to not execute.
    WARNING: Executing the EXIT command will make it
    necessary to start R:base again. Are
    you really finished with your R:base
    session?

ENDC
```

To see this prompt block on the screen, at the R> prompt type:

```
PROMPT EXIT
```

A display that looks like this appears on the screen:



Prompt blocks may

- Contain parameters in the command line which are filled in by responses to prompt lines by the operator
- Cause other prompt blocks to be displayed

### PROMPT BLOCKS WITH PARAMETERS

Below is a prompt block from *prompt.dat* which prompts the operator for input in order to complete the command in the command line of the prompt:

```

USER
USER @1
    The USER command gives you an alternative to the SET USER command for
    setting your current user password.

    Enter your user password : @1
ENDC

```

To display the prompt screen, at the R> prompt type:

```
PROMPT USER
```

A screen like this is displayed:

PROMPTS	
<b>USER</b>	
The USER command gives you an alternative to the SET USER command for setting your current user password.	
Enter your user password :	<input type="text"/>

When the operator enters a password in the reverse video field following the prompt line and presses [ESC], the entered password is substituted for the @1 in the command in line two of the prompt block.

You can prompt the operator for up to ten items to create the command line using @1, @2, @3, . . . , @0. The maximum length for any one prompted entry is 78 characters.

If an item is not filled in on the prompt screen, nothing is included in the command line corresponding to the missing item. This enables you to have optional parameters in the prompt screen. Everything after the last parameter filled is omitted from the command line that is executed.

For example, the prompt block shown below uses an optional parameter:

```

COMPUTE
COMPUTE @3 @2 FROM @1 WHERE @4
  Use COMPUTE to calculate either the count, minimum, maximum, average, sum, rows or
  all of these values for the column you specify.

  Name of table that contains column           : @1
  Name of column that contains values          : @2
  SUM, AVG, COUNT, MAX, MIN, ALL or ROWS      : @3
  Condition with which to select rows (optional) : @4
ENDC

```

If the operator does not fill in @4, the command is composed as

```
COMPUTE @3 @2 FROM @1
```

**PROMPT BLOCKS THAT DISPLAY OTHER PROMPT BLOCKS**

The following example shows a prompt block that causes another prompt block to be displayed:

```

EDIT
PROMPT ED@1
    Use EDIT to access a table for browsing through rows, changing
    or updating data, or deleting rows.

    Which display do you want?
    1—Customized data-entry form (choose only if a form is defined
                                for the table you want to edit)
    2—Standard RBASE format
    Enter a choice (1 or 2): @1
ENDC

```

The above prompt block executes the command

```
PROMPT ED1
```

if the operator enters *1* at the prompt line. If the operator enters *2*, the prompt block executes the command

```
PROMPT ED2
```

For this to work, there must be two more prompt blocks name *ED1* and *ED2*. When composing linked prompt blocks like this, realize that only the first six characters of a prompt block name are significant. Prompt blocks named *select1* and *select2* in *prompt.dat* cannot be differentiated and only the first one would be placed in *prompt.rbs* when *promgen.exe* is executed.

Following are examples of two prompt blocks *ED1* and *ED2*:

```

ED1
EDIT USING @1 SORTED BY @2 WHERE @3
    Form name of the customized form                : @1
    Sort order (optional list of column names)       : @2
    Conditions which select rows (optional)          : @3
ENDC
ED2
EDIT @2 FROM @1 SORTED BY @3 WHERE @4
    Name of table to edit                            : @1
    Name(s) of columns to edit (or ALL)              : @2
    Sort order (optional list of column names)       : @3
    Conditions which select rows (optional)          : @4
ENDC

```



## Generating a Prompt File

Make sure your new *prompt.rbs* file is on the default drive. Run *promgen.exe* which automatically reads *prompt.dat* as input and generates the new *prompt.rbs* file on the default drive.

## VARIABLE FORMS

Variable forms differ from table forms in these ways:

- Values entered or edited in a variable form can be moved into more than one table.
- During data editing, values can be displayed on a variable form which the operator cannot edit.
- Calculations can be done on values entered through a variable form. The results can be moved into the database, or displayed on the variable form where the operator can edit them before they are moved to the database.
- When the operator is filling in a variable form, values entered into the form can be used to look up in tables the values of other variables. These values can be displayed in the variable form.
- Variable forms may include several screens.

## The FORMS Command

To begin making a variable form, at the R> prompts type: *FORMS*

Press [ENTER] when you are prompted for a table name to specify that you want to define a variables form.

A variable form includes a title, labels, and fields. The title describes the purpose of the form. Each label describes a variable. Each field is displayed in reverse video. The operator may have entry or edit access to the field. The current value of the variable that corresponds to the field may be displayed. An example variable form is shown in two representations. The first shows the form as it appears on the screen. Figure 15-4 shows the same form in the planning stage as a design sketch on paper. The sketch shows the name of the variable that stands behind each data entry field.

SAMPLE COMPANY NAME  
CLIENT RECORD FORM

Customer number:  Employer:

Contact:  
First Name:  Initial:  Last Name:   
Address:

City:  State:  Zip code:

Phone:

Credit terms:

SAMPLE COMPANY NAME  
CLIENT RECORD FORM

Customer Number: Cust# Employer: CoName

Contact:  
First Name: Frstname Initial: MI Last Name: LName  
Address: Addr 1  
Addr 2

City: City State: Statecd Zip Code: Zipcode

Phone: AC Phone

Credit Terms: CredTerm

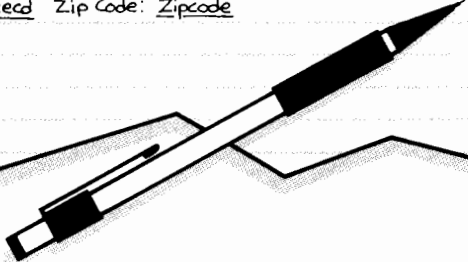


Figure 15-4 A Variable Form

Figure 15-5 suggests a way to think of the relationship between a variable form and the variables. The variables are behind the form. When the operator enters a value, that value goes into the variable. In figure 15-4, the field *Employer* has this relationship to the variable *CoName*. Behind the variables are the database columns to which variables are linked. After the variable form is defined using the FORMS command and the definition is stored in the database FORMS table, the DRAW, ENTER VAR, and EDIT VAR commands are used to manipulate it.

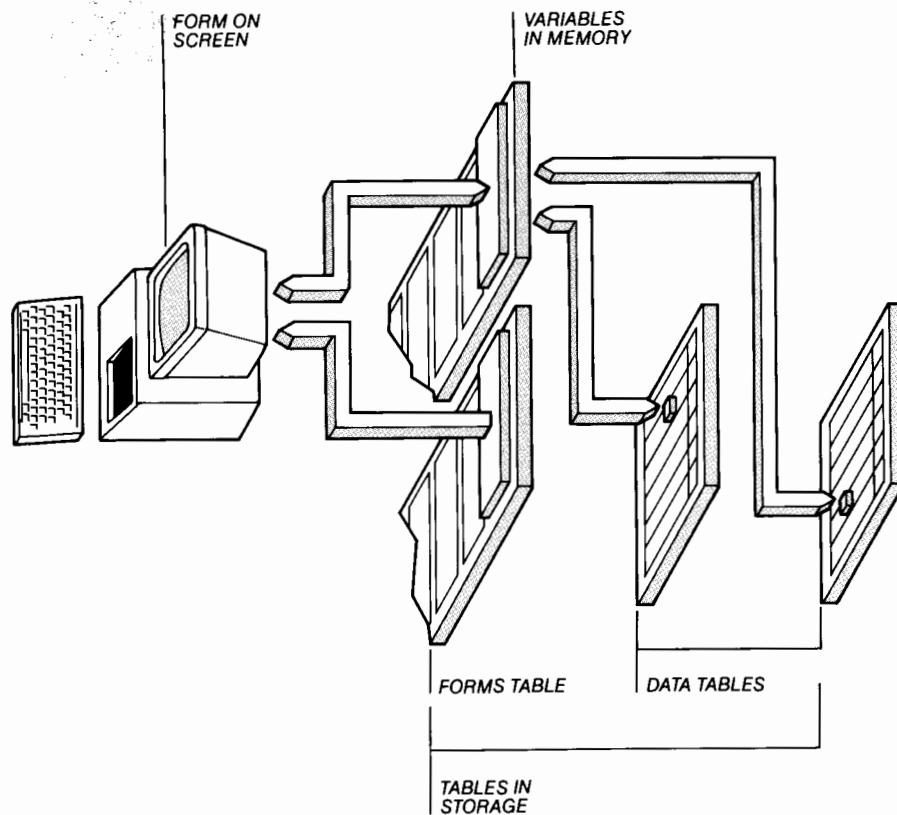



Figure 15-5 Relationships Between Variables Form, Variables, and Tables

## The DRAW Command

Use the DRAW command to draw a variable form on the screen. Up to five variable forms can be drawn on the screen at a time.

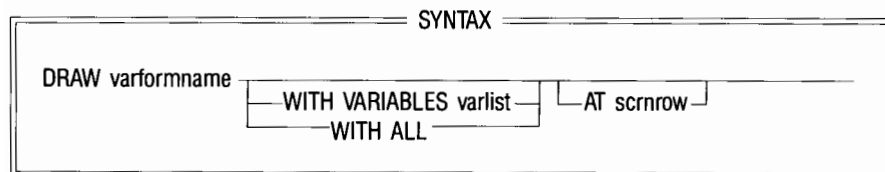
The DRAW command can be used in four different ways, each corresponding to a different option of the command. These command options and functions are shown in the following table. The DRAW command options in table 15-4 manipulate the variable form shown in figure 15-4.

Table 15-4 DRAW Command Options and Functions



Option	Function
DRAW newcus	Displays the text from the form definition, starting on line 2 of the screen
DRAW newcus AT 12	Displays the text from the form definition, starting at line 12 on the screen
DRAW newcus WITH ALL	Displays the text from the form definition and shows the value of the variable in each field
DRAW newcus WITH Cust# CoName	Displays the text from the form definition but shows the current value of only the variables <i>Cust#</i> and <i>CoName</i>

The AT clause can be used with the latter two DRAW commands listed above to place the form on the screen. The syntax of the DRAW command is:



As many as five variable forms can be displayed on the screen with the DRAW command before you must issue a NEWPAGE command.

## The ENTER VAR Command

Use the ENTER VAR command to specify the fields into which an operator may enter values. The ENTER VAR command uses the four options listed in table 15-5.

Table 15-5 ENTER VAR Command Options and Functions

Option	Function
ENTER VAR	Operator can enter data into all fields on the currently active form.
ENTER VAR Frstname Lname*	Operator can enter data into the fields for variables <i>Frstname</i> and <i>Lname</i> .
ENTER VAR USING newcus	Operator can enter data into all fields in the form named <i>newcus</i> . If <i>newcus</i> is already drawn on the screen, then the fields will be placed within the <i>newcus</i> text. Otherwise, the fields will be placed on the screen without a text mask, starting at line 1 of the screen, which may have the text from another variable form drawn there. This enables different sets of variable values to be entered using the same text mask.
ENTER VAR RETURN + ENTER ESC PGDN PGUP	Operator can enter data into all fields in the currently active form. Operator can use [ENTER], [ESC], [PgUp], and [PgDn] when finished working with this form and return control to the command file. The name of the key pressed will be in a variable named #RETURN.

\*The fields the operator can enter are limited to those listed after the keyword VAR. If none are listed, all can be edited.

The syntax of the ENTER VAR command is:

SYNTAX			
ENTER VARIABLE	[ varlist ]	USING varformname	[ RETURN keylist ]

## The EDIT VAR Command

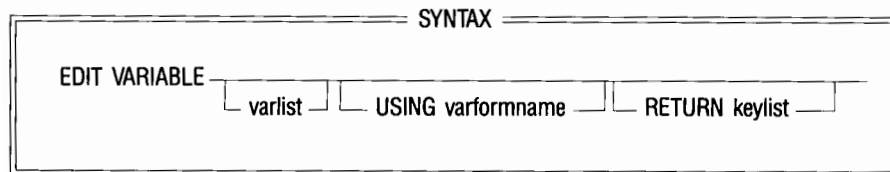
Use the EDIT VAR command to specify the fields on a variable form in which the current value of a variable may be displayed so the operator may see the value and change it if necessary. The EDIT VAR command uses the four options listed in table 15-6.

Table 15-6 EDIT VAR Command Options and Functions

Option	Function
EDIT VAR	Operator can edit values in all fields on the currently active form.
EDIT VAR Frstname Lname*	Operator can only edit values in the fields for variables <i>Frstname</i> and <i>Lname</i> .
EDIT VAR USING newcus	Operator can edit values in all fields in the form named <i>newcus</i> . If <i>newcus</i> is one of the variable forms drawn on the screen, then the fields will be placed within the <i>newcus</i> text. Otherwise, the fields will be placed on the screen without text, starting at line 1 of the screen, which may have the text from another variable form drawn there. This enables different sets of variable values to be entered using the same text mask.
EDIT VAR RETURN + ENTER ESC PGDN PGUP	Operator can edit values in all fields in the currently active form. Operator can press [ENTER] [ESC], [PgUp], and [PgDn] when finished working with this form and return control to the command file. The name of the key pressed will be in a variable named #RETURN.

\*The fields the operator can edit are limited to those listed after the keyword VAR. If none are listed, all can be edited.

The syntax of the EDIT VAR command is:



## Command File Examples

Following are four command file segments that demonstrate how the DRAW, ENTER VAR, and EDIT VAR commands can be used together.

### FROM PAPER FORM TO MULTIPLE DATABASE TABLES

Figure 15-6 shows a form on paper with data that is to be stored in several tables.

CLIENT INFORMATION	
ID:	_____
Name:	_____
Family Size:	_____
Family Income:	_____
HOUSING INFORMATION	
Address	_____
Type:	_____
Number of Rooms:	_____
GRANT INFORMATION	
Energy:	_____
Weatherization:	_____
Construction:	_____

Figure 15-6 A Paper Form

The information is on one paper form, but in the database the data is stored in the tables *clients*, *houses*, and *grants*. The client identification number is the common column that links the three tables. The following command file segment allows an operator to enter data into one variable form. The command file moves the entered data into the appropriate tables.

```

*(Moving data from one form to several tables)
SET VAR more TO "Y"
WHILE more EQ Y THEN
  NEWPAGE
  DRAW vf1 WITH ALL AT 7
  ENTER VAR
  LOAD clients
    .vidno .vname .vfamily .vincome
  END
  LOAD houses
    .vidno .vaddress .vtype .vrooms
  END
  LOAD grants
    .vidno .venergy .vweather .vcontru
  END
  CLEAR ALL VARIABLES
  FILLIN more USING "Another client to enter (Y/N)?" AT 24 10
ENDWHILE

```

The variables loaded into the *clients*, *houses*, and *grants* tables (*vidno*, *vfamily*, *vincome*, and so on), are defined in the variable form named *vf1*. Once a value is entered via the form into these variables, the values can then be loaded into the correct table as shown in the three *LOAD* command segments.



**DATA INTEGRITY AND SECURITY**

Figure 15-7 shows a paper consumer-credit form used to record address changes. The salary and credit rating fields are not filled in. The programming task is to build a command file that enables the operator to edit the address of a client. You want the operator to see the ID# and name from the client's record but not be able to edit them. The operator should not see the salary or credit rating.

I.D. No.:	_____
First Name:	_____
Last Name:	_____
Address:	_____
City:	_____
State:	_____
Zip:	_____
Salary:	_____
Credit Rating:	_____

Figure 15-7 Sample Consumer Credit Form

You have a variable form, named *vf1*, stored in the database which has text and fields defined for all the items on the paper form.

The DRAW command can be used to display the current values of all variables corresponding to form items except the salary and credit rating. The EDIT VAR command limits the operator to editing only address information in the example below:

```
*(Limiting the variables the operator sees and can edit)
SET MESSAGES OFF
SET ERROR MESSAGES OFF
SET VAR vidno TO 0
NEWPAGE
```

```

WHILE vidno EXISTS THEN
  FILLIN vidno USING "Enter client ID number: (Press [ENTER] to quit):" AT 1 1
  IF vidno FAILS THEN
    SET MESSAGES ON
    SET ERROR MESSAGES ON
    BREAK *(QUIT, INPUT TERMINAL, or GOTO getout command here gives same +
    result)
  ENDIF
  SET POINTER #3 e3 FOR cusrec WHERE idno = .vidno
  IF e3 EQ 0 THEN
    SET VAR vfirst TO FrstName IN #3
    SET VAR vlast TO LastName IN #3
    SET VAR vaddress TO Address IN #3
    SET VAR vcity TO City IN #3
    SET VAR vstate TO State IN #3
    SET VAR vzip TO Zip IN #3
    DRAW vf1 WITH VAR vidno vfirst vlast vaddress vcity vstate vzip
    EDIT VAR vaddress vcity vstate vzip
    IF #RETURN EQ ESC THEN
      GOTO getout
    ENDIF
    CHANGE Address TO .vaddress IN #3
    CHANGE City TO .vcity IN #3
    CHANGE State TO .vstate IN #3
    CHANGE Zip TO .vzip IN #3
  ELSE
    FILLIN cont USING "Client ID not on file-press [ENTER] to +
    continue" AT 3 1
  ENDIF
  LABEL getout
  NEWPAGE
ENDWHILE
RETURN

```

### TABLE LOOKUP AND DOING CALCULATIONS

Suppose your firm has implemented a five percent raise to most employees. Specifically, you want to:

- Enter an employee's first name, last name, and ID number.
- Have the command file use that information to look up the employee's position, years of service, and salary.
- Have the command file calculate a five percent raise on the current salary.
- Have the command file display the employee's position, years, and calculated salary, keeping the name and ID number information on the screen at the same time.

- Have the command file give you editing access to the displayed new salary so you can raise or lower it. You can signal the command file that you are finished editing by pressing [ENTER].
- When you finish with an employee's record, the command file will use the information in the variable to add a row to a table named newsal and enable you to enter another employee's ID number to repeat the process.

The following segment is from a command file that can do this.

```
*(Doing table lookup and calculations with variable forms)
NEWPAGE
SET MESSAGES OFF
SET VAR idno TO 0
WHILE idno EXISTS THEN
  DRAW vf1 AT 2
  ENTER VAR idno RETURN ENTER ESC
  IF #RETURN EQ ESC THEN
    BREAK
  ENDIF
  SET POINTER #3 e3 FOR empfile WHERE id = .idno
  IF e3 EQ 0 THEN
    SET VAR name1 TO name1 IN #3
    SET VAR name2 TO name2 IN #3
    DRAW vf1 WITH ALL AT 2
    SET VAR posit TO posit IN #3
    SET VAR years TO years IN #3
    SET VAR currsal TO salary IN #3
    SET VAR newsal TO 105 % .currsal
    DRAW vf2 WITH VAR posit years AT 6
    EDIT VAR newsal USING vf2 RETURN ESC ENTER
    IF #RETURN EQ ENTER THEN
      SET VAR currsal TO .newsal
      LOAD newsal
      .idno .name1 .name2 .posit .years .currsal
    END
  ENDIF
  NEWPAGE
ENDIF
ENDWHILE
NEWPAGE
RETURN
```

Figure 15-8 shows the relationships between the variable forms *vf1* and *vf2*, drawn on the screen, and the tables *empfile* and *newsal*. (a) shows variable form *vf1* drawn on the screen, the operator entering an I.D. No. into the form, and *idno* used to select the table row which initializes the variables. (b) shows both forms *vf1* and *vf2* drawn including the variable values. The operator can edit the *NEW SALARY* field. (c) shows the variable values loaded into *newsal* table.

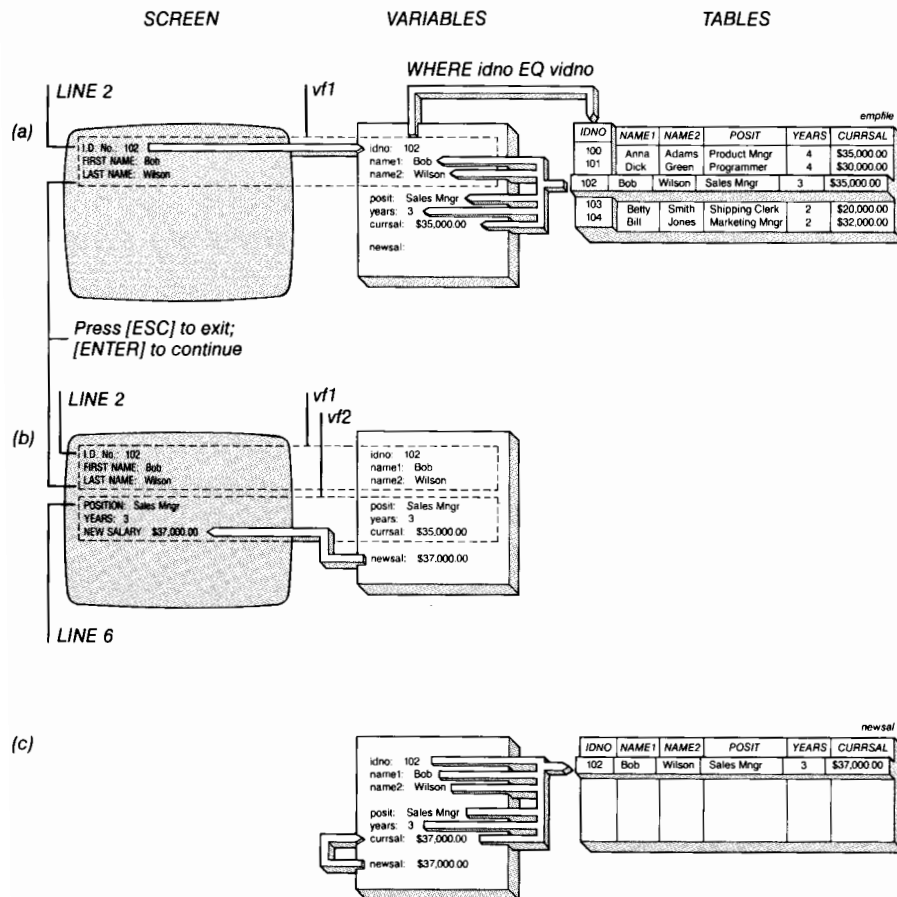


Figure 15-8 Relationships Between Variable Forms *vf1* and *vf2*, the Screen, and the Tables

**MULTIPLE PAGE FORM EMULATION**

Suppose you have a three-page paper form that contains the values to be entered into your database. The keylist feature of the ENTER VAR command can be used to construct a three-page electronic form. The following command file segment does this.

```
*(Emulating multiple page input forms)
NEWPAGE
CLEAR ALL VARIABLES
SET NULL " "
SET VAR pg TO 1
*(suppress system prompt for forms in line so you can use own prompt line)
SET MESSAGES OFF
WHILE pg GT 0 THEN
  NEWPAGE
  WRITE "Press [ESC] to quit, [PGUP] to move up, [PGDN] to move down +
  a page" AT 1 1
  IF pg EQ 1 THEN
    DRAW vf1
  ENDIF
  IF pg EQ 2 THEN
    DRAW vf2
  ENDIF
  IF pg EQ 3 THEN
    DRAW vf3
  ENDIF
  EDIT VAR RETURN ESC PGUP PGDN
  IF #RETURN EQ PGUP THEN
    IF pg GT 1 THEN
      SET VAR pg TO .pg - 1
    ELSE
      SET VAR pg TO 3
    ENDIF
  ENDIF
  IF #RETURN EQ PGDN THEN
    IF pg LT 3 THEN
      SET VAR pg TO .pg + 1
    ELSE
      SET VAR pg TO 1
    ENDIF
  ENDIF
ENDIF
```

```

IF #RETURN EQ ESC THEN
  LOAD tblname
    .v1 .v2 .v3 .v4 .v5 .v6 .v7 .v8 .v9 .v10 .v11 .v12
  END
  NEWPAGE
  WRITE "Enter another row? (Y)" AT 12 25
  FILLIN cont USING "(" AT 12 44
  IF cont FAILS OR cont EQ Y THEN
    CLEAR ALL VARIABLES
    SET VAR pg TO 1
  ELSE
    SET MESSAGES ON
    SET NULL -0-
    BREAK
  ENDIF
ENDIF
ENDWHILE
RETURN

```

## DATABASE ACCESS

The R:Base 5000 programming language has an especially useful feature for accessing and updating data in your database that will be the foundation of almost all of the R:Base programs you write. This feature is the SET POINTER and NEXT commands.

There are two principal items to keep in mind about SET POINTER and NEXT; these can be called buffer and route. You can set up three different buffers or data pages for your database at the same time. Each buffer holds rows of data from one table. The size of this internal R:base buffer is 1530 bytes. This buffer is set up by the SET POINTER command. SET POINTER provides the table name, the selection criteria for the rows (WHERE clause), and the order in which they are to be accessed (SORTED BY).

The set of rows selected and their order establish a route or pathway through the table. Each disk read gets enough rows to fill up the 1530 byte buffer. The SET POINTER command points to the first row in the route through the table. The NEXT command advances the route pointer by one row. Since there are three buffers available, three tables can be addressed efficiently at the same time. If a lookup to the database is done in an R:base program without explicitly using a SET POINTER route, the system uses the buffer for route #1. It is therefore recommended that route #2 or #3 be used as the primary one if a non-SET POINTER access is also to be used in your program.

There are numerous excellent models of the use of SET POINTER in the sample command file segments provided in the command file examples used in this chapter. In order to gain skill in programming in R:base, study these sample programs and imitate their style. Programming skill is a matter of practice.

If SET POINTER and NEXT make up the foundation, then the cornerstone is the route number (#n) status variable. The status variable is *e3* in the sample below.

```
SET POINTER #3 e3 FOR table
WHILE e3 = 0 THEN
  SET VAR v1 TO col1 IN #3
  SHOW VAR v1
NEXT #3 e3
ENDWHILE
```

If no rows are found, *e3* will equal 2406. When there are no more rows to find, *e3* will equal 2406. If a row is found, *e3* will equal zero (0). This example finds all the rows in the table, and for each row, it prints the value of the column *col1* on the screen.

## MESSAGES AND ERRORS

R:base has both administrative (or diagnostic) messages and error messages. Either or both can be suppressed or turned back on with these commands:

```
SET MESSAGES OFF
SET ERROR MESSAGES OFF
SET MESSAGES ON
SET ERROR MESSAGES ON
```

Some errors can be trapped using the ERROR VAR you define. The error variable, if defined, is set after each command is executed. If no error condition occurs, the ERROR VAR is set to zero (0). If an error is found, the ERROR VAR is set to a non-zero value. A list of error messages and their corresponding error values is given in the *R:base 5000 Reference Manual*.

```
SET ERROR VAR varname
SET ERROR VAR OFF
```

To make use of error trapping, you need to execute a control statement before any further processing takes place. To do this you must test the value of the error variable (and if desirable or necessary put its value into another variable) before it is changed by another command. Only the R> mode and the COMMAND file mode allow the necessary trapping statements. Errors cannot be directly trapped in the LOAD L> mode or in the DEFINE D> mode. An error resulting from using an undefined variable cannot be trapped. An error message is displayed even when the SET ERROR MESSAGES OFF command has been issued.

All errors that set the ERROR VAR to a non-zero value also put an error message into the error message buffer. The contents of this buffer can be printed using this command:

```
SHOW ERROR varname AT scrnrow scrncol
```

The following set of commands prompt the operator for a database name without terminating the command file. It shows how you can trap error messages and display them on the screen.

```
*(Sample error handler for OPEN command)
SET MESSAGES OFF
SET ERROR MESSAGES OFF
SET ERROR VAR ev
SET VAR space TO " "
SET VAR k TO 0
WHILE k GE 0 AND k LT 10 THEN
  NEWPAGE
  FILLIN db USING "Enter name of database to open:" AT 4 5
  OPEN .db
  IF ev NE 0 THEN
    SET VAR k TO .k + 1
    IF k = 10 THEN
      WRITE "Sorry, tried 10 times to open database, you're out" *(Your error +
        message)
      GOTO xend
    ENDIF
    WRITE "Unable to open database" AT 5 5
    SHOW VAR space AT 9 1
    WRITE "Press any key to continue" AT 9 5
    PAUSE
  ELSE
    WRITE "Database opened" AT 5 5
    BREAK *(Success)
  ENDIF
ENDWHILE
LABEL xend
SET MESSAGES ON
SET ERROR MESSAGES ON
```



## DEBUGGING

Debugging a command file essentially involves running the command file and tracing its execution. There are three basic techniques involved.

You can run the command file and display intermediate variable values using the SHOW VARIABLES and PAUSE commands. You can also run the command file and print intermediate results to an output file (OUTPUT *filename* WITH SCREEN).

When the command file is finished executing, you can study the results. Use the ECHO command to observe what happens, line by line, as the program executes. ECHO will print out each command as it is encountered. If ECHO is on, any variables inside a WHILE loop are displayed with the value of the variable substituted for the variable name.

These three techniques can be used together if desired.

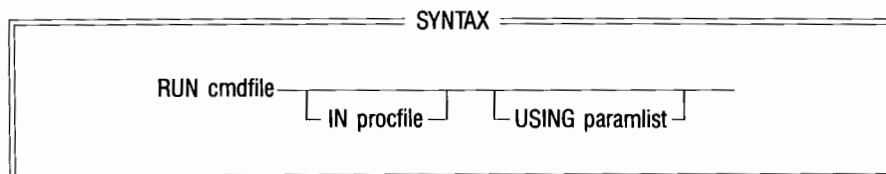
## APPLICATION INTEGRATION

### Subroutines

When one COMMAND file calls another with INPUT or RUN, the called file is referred to as a subroutine. When the RUN command is executed, values in the calling program are passed to the subroutine. The values can be variables that are independent of the calling program. This capability is useful when the subroutine does a formal process that can be used over and over again by different programs. This process is often called passing parameters or passing arguments.

There are several sample subroutines included with R:base 5000 that use parameter passing. See "Macros" in this chapter for a detailed list.

The RUN command can pass up to nine (9) parameters to the called subroutine. The syntax for the RUN command is:



*Paramlist* is the parameter list.

Parameters can be constants or dotted variables. Inside the called subroutine, the passed parameters are held in special system variables of the form:

```

.%1 .%2 ... .%9

```

These substitution variables take on the value given in the RUN command parameter list. Do not use the null symbol -0- unless the subroutine has some provision for recognizing it as a null value.

R:base stores these variables in its variable table. A suffix representing the nesting level of the subroutine is appended to these variables. They can be examined by SHOW VAR. In the following example, a file two levels down (*two.cmd*) calls a subroutine (*my.sub*) and passes three parameters to it. The code looks like this:

```

*( TWO.CMD )                (This is a 2nd level file)
. . . .
RUN my.sub USING .A .B .C
. . . .
*( EOF )

*( my.sub name rank serial# ) (This is a 3rd level file)
. . . .
SET VAR name TO .%1
SET VAR rank TO .%2
SET VAR serial TO .%3
. . . .
RETURN
. . . .
*( EOF )

```



The variable table would have something in it like this:

A	JOHNNY JONES
B	CAPT
C	123456
%1-3	JOHNNY JONES
%2-3	CAPT
%3-3	123456

Good programming practice calls for a RETURN command at the end of every subroutine. Do not, however, use a RETURN command within an IF...THEN structure or a WHILE loop.

## Macros

R:base 5000 includes six macros that are ready for you to use. The macros are listed in table 15-7, and an explanation of each is in the *macro.doc* file.

Table 15-7 R:base 5000 Macros

Macro Name	Purpose
BACK.MAC	Backs up a database
DELFR.MAC	Deletes forms and reports from a database
FREQ.MAC	Calculates subtotals and percentages
LABELS.MAC	Prints mailing labels
POST.MAC	Posts accounting transactions in a general ledger file
SUMM.MAC	Calculates subtotals and totals sorted by transaction type

To use these macros, enter a RUN command with the parameters specific to your application filled in. For additional information on passing parameters see the "Subroutines" section. You cannot use these macros directly in an application built with the EXPRESS. To use them with an application you must select the *Custom* feature of the EXPRESS to build a command block. The command block would include a RUN command containing the proper parameters to be passed to the selected macro. If desired, you can also collect the parameter information from the operator to pass to the macro. For example, the custom code you would enter to use the back-up macro might look like this:

```
SET VAR drive TEXT
SET VAR dir TEXT
SET VAR file TEXT
FILLIN drive USING "Enter the drive: "
FILLIN dir USING "Enter the directory: "
FILLIN file USING "Enter the back up file: "
RUN back.mac USING .drive .dir .file
RETURN
```

## The Initial Command File

When R:base is loaded, it looks for a file named *rbase.dat* in the current directory on the default drive. If it finds such a file, it begins processing the commands in this file.

This file is useful for ensuring that operators of your customized application enter all the necessary information, including their passwords, and carry out all the steps in the application in the same way every day.

To use an initialization file with a hard disk system, you need an *r:base.dat* file for each database and application, if you place each application on a separate directory and also place its *rbase.dat* file there.

On a floppy disk system, place the *rbase.dat* file on the application and database disk in drive *b*:

You make an initial command file the same way you make any command file—by using RBEDIT or some other editor. When you save the file, name it *rbase.dat*. Store it as described above.

The EXPRESS offers the option of building an *rbase.dat* file for you. If you request this option, the application you are building with EXPRESS is automatically executed when you load R:base. The EXPRESS *rbase.dat* file looks something like this:

```
RUN appname IN appfile.apx  
EXIT
```

Notice that it has an EXIT command which is used to leave R:base completely when the operator exits from the application.



---

## RCOMPILE Contents

<b>How to Use This Chapter</b>	16-2
<b>RCOMPILE Overview</b>	16-2
<b>RCOMPILE Main Menu Options</b>	16-5
Converting an ASCII Command File to a Binary Command File	16-5
Adding an ASCII Command File to a Procedure File	16-7
Adding an ASCII Screen File to a Procedure File	16-8
Adding an ASCII Menu File to a Procedure File	16-9
Converting an ASCII Application File to a Binary Application File	16-11
Displaying the File Directory	16-13
Displaying the Contents of an ASCII File	16-14
Exiting RCOMPILE	16-14

## HOW TO USE THIS CHAPTER

This chapter explains how to use RCOMPILE, the R:base 5000 compiler, to convert ASCII files to a binary format. RCOMPILE accepts as input ASCII files that are created with the Application Express, RBEDIT (R:base 4000 or R:base 5000), or any text editor. See chapter 15 for an explanation of the R:base 5000 programming language and sample applications.

## RCOMPILE OVERVIEW

You can use RCOMPILE to accomplish the following conversions:

- ASCII command file to a binary command file
- ASCII command file to a command block, and add the block to a procedure file
- ASCII screen file to a screen block, and add the block to a procedure file
- ASCII menu file to a menu block, and add the block to a procedure file
- ASCII application file to a binary application file

Figure 16-1 shows the input and output file types that are compatible with RCOMPILE.

RCOMPILE accepts ASCII command, menu, screen, and application files as input.

- Command files contain R:base commands. These files may be compiled independently using option 1 or added to an existing procedure file using option 2.
- Menu files contain the information and structure used by the R:base CHOOSE command to display a menu on the screen. These files may be compiled as an independent binary file or added to an existing procedure file using option 4.
- Screen files contain the information and structure used by the R:base DISPLAY and TYPE commands to display prompts on the screen. These files may be compiled as an independent binary file or added to an existing procedure file using option 3.

Application files contain command, menu, and screen files separated into blocks of code. Each application file may contain up to 42 blocks, or subroutines. There are three types of blocks:

- Command blocks are complete command files contained within an application file and are defined by the word *\$COMMAND* appearing as the first line of code. You enter the *\$COMMAND* line only if you are compiling an entire application file using option 5.
- Menu blocks are menu definitions contained within an application file and are defined by the word *\$MENU* appearing as the first line of the definition.
- Screen blocks are screen display definitions contained within an application file and are defined by the word *\$SCREEN* as the first line of the definition.

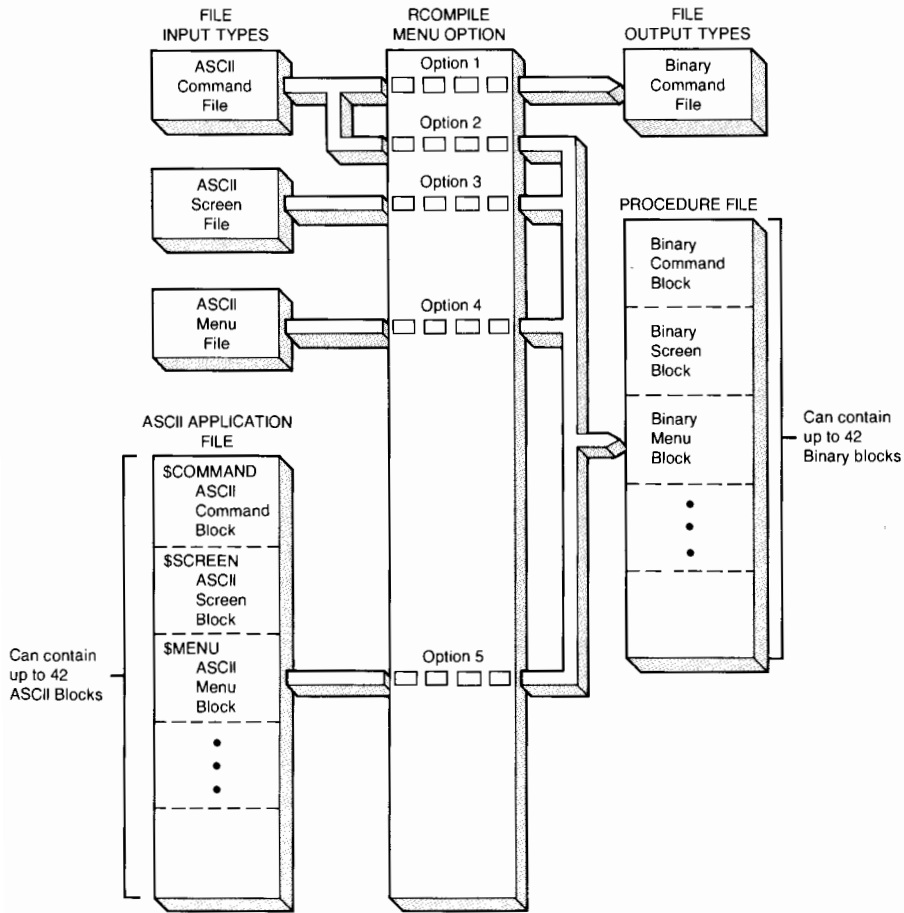


Figure 16-1 RCOMPILE Input and Output File Types



There is no mandatory order that the blocks must conform to when they are joined in an application, but applications are most efficient if frequently used blocks are located first in the file. The compiler automatically defines command, menu, and screen blocks when you use options 2, 3, and 4 to add one of these block types to an application file. Do not include *\$COMMAND*, *\$MENU*, or *\$SCREEN* as the first line of these files. If you want to code an entire application file containing all of the command, menu, and screen blocks you need, you must separate each block with the appropriate *\$* word before compiling it using option 5.

As shown in figure 16-1, the four types of ASCII files can be converted to binary files using various options of RCOMPILE. Option 1 produces a simple binary command file containing only R:base commands. The other options are all used to add various types of command blocks to a procedure file. Options 2, 3, and 4 add command, menu, and screen blocks while option 5 produces the complete procedure file with all command blocks required for operation.

There are three distinct advantages to using files that have been converted with RCOMPILE instead of simply using ASCII files. First, the code for the application is more secure. This is because the file is in binary form, instead of English, and therefore cannot be read or changed with a text editor.

Second, the execution time for the file may be reduced. This is because:

- The command lines have already been partially interpreted by RCOMPILE. As a result, some of the work the computer needs to do to execute the ASCII form of the command file is not necessary.
- Calls between binary blocks in procedure files can be accomplished without taking the time to open and close a different DOS disk file for every call.

Third, command blocks in procedure files require less code to call other command blocks. This is because the list of parameters with a RUN command can be passed to the called block. See the description of the RUN command in the *R:base 5000 Reference Manual* "Command Dictionary" and in chapter 15.

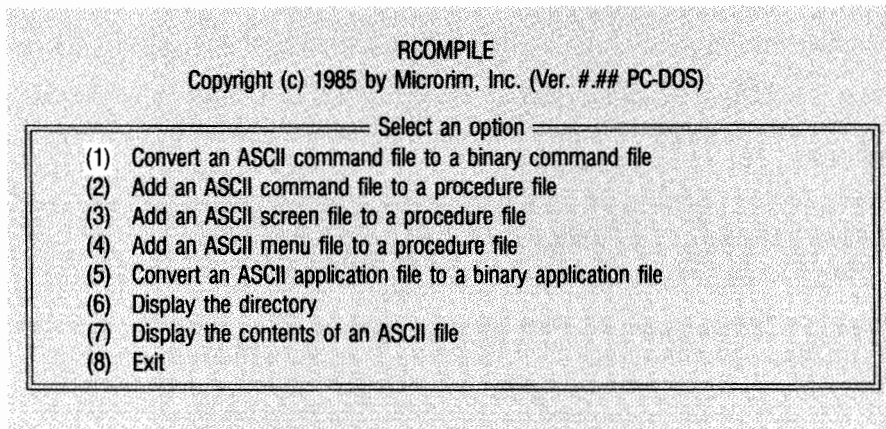
Before you submit a command file to RCOMPILE as input, make sure that your file does not contain any non-ASCII characters. Then test the file to make sure it contains no errors. RCOMPILE does not check the validity of commands. See chapter 15 for a procedure to test R:base command files.

## RCOMPILE MAIN MENU OPTIONS

To run RCOMPILE, select *RCOMPILE* from the R:base 5000 main menu, or at the DOS prompt, enter:

RCOMPILE

The RCOMPILE main menu is shown below. Select the number of the option you want, or move the cursor to the option and press [ENTER].



### Converting an ASCII Command File to a Binary Command File

Select option 1 from the RCOMPILE main menu to convert an ASCII command file to a binary command file. The main menu remains on the screen, and RCOMPILE responds with the following prompt:

Name of the ASCII command file to convert:

Enter the name of the file you want to convert. If the file is not located on the default drive and directory, be sure to include the drive and path specification before the file name. For example, if the file *startup1.cmd* is located on drive *c:* in the directory *comfile*, enter:

```
c:\comfile\startup1.cmd
```

R\_COMPILE responds with the following message:

```
Name of the back-up file [c:\comfile\startup1.asc]:
```

Press [ENTER] to assign the name shown in brackets to the back-up file for the original file, or assign a new name. R\_COMPILE responds with the following message:

```
Name of the binary command file [c:\comfile\startup1.com]:
```

Press [ENTER] to assign the name shown in brackets to the binary file, or assign a new name. After the binary file name is assigned, R\_COMPILE generates the binary file. As each command in the command file is converted to binary form, the ASCII command is displayed on the screen. The following screen shows the three R\_COMPILE prompt lines and the result after the *startup1.cmd* file is processed. When the conversion is complete, you are prompted to press any key on the keyboard to continue.

```
Name of the file to convert:c:\comfile\startup1.cmd
Name of the back-up file [c:\comfile\startup1.asc]:
Name of the binary command file [c:\comfile\startup1.com]:
SET ECHO ON
OUTPUT c:session.day WITH TERMINAL
SET BELL OFF
OPEN c:compuco
QUIT

Press any key to continue
```

Once converted, the compiled command file is executed in R:base by the following command:

**RUN filespec**

*Filespec* is the full name of the binary command file including drive and path specifications if the file is not on the current drive and directory.

## Adding an ASCII Command File to a Procedure File

Select option 2 from the RCOMPILE main menu to convert an ASCII command file to a binary command block and add the block to a procedure file. The main menu remains on the screen, and RCOMPILE responds with the following message:

Name of the ASCII command file to add:

Enter the name of the file you want to convert. If the file is not located on the default drive and directory, be sure to include the drive and path specification before the file name. For example, if the file *startup2.cmd* is located on drive *c:* in the directory *comfile*, enter:

`c:\comfile\startup2.cmd`

RCOMPILE responds with the following message:



Name of the procedure file [c:\comfile\compuco2.prc]:

Press [ENTER] to assign the name shown in brackets to the file, or if no default file exists, enter the name of the new or existing procedure file to which you want to add the converted file. After you press [ENTER], RCOMPILE responds with the following message:

Name of the binary command block [startup2]:

Press [ENTER] to assign the name shown in brackets to the binary block, or assign a new name. After the block name is assigned, RCOMPILE generates the binary file. As each command in the command file is converted to binary form, the ASCII command is displayed on the screen. The screen below shows the three message lines and the result after the *startup2.cmd* file is added. When the addition is complete, you are prompted to press any key on the keyboard to continue.

Name of the ASCII command file to add: c:\comfile\startup2.cmd

Name of the procedure file [c:\comfile\compuco2.prc]:

Name of the binary command block [startup2]:

```
SET ECHO ON
OUTPUT PRINTER WITH TERMINAL
SET BELL OFF
OPEN c:compuco
QUIT
```

Press any key to continue

The specific command block in the procedure file to which the command block is added is executed in R:base by the following command:

```
RUN blockname IN filespec
```

*Blockname* is the name of the command block and *filespec* is the full name of the binary procedure file including drive and path specifications if the file is not on the current drive and directory.

### **Adding an ASCII Screen File to a Procedure File**

Select option 3 from the RCOMPILE main menu to convert an ASCII screen file to a binary screen block and add the block to a procedure file. The main menu remains on the screen, and RCOMPILE responds with the following message:

Name of the ASCII screen file to add:

Enter the name of the file you want to convert. If the file is not located on the default drive and directory, be sure to include the drive and path specification before the file name. For example, if the file *mainhelp.scr* is located on drive *c:* in the directory *comfile*, enter:

```
c:\comfile\mainhelp.scr
```

RCOMPILE responds with the following message:

Name of the procedure file [c:\comfile\compuco1.prc]:

Press [ENTER] to assign the name shown in brackets to the file, or if no default file exists, enter the name of the new or existing procedure file to which you want to add the converted file. After you press [ENTER], RCOMPILE responds with the following message:

Name of the binary screen block [mainhelp]:

Press [ENTER] to assign the name shown in brackets to the binary block, or assign a new name. After the block name is assigned, RCOMPILE generates the binary file. As each command in the command file is converted to binary form, the ASCII command is displayed on the screen. The screen below shows the three message lines and the result after *mainhelp.scr* is added. When the addition is complete, you are prompted to press any key on the keyboard to continue.

```
Name of the ASCII screen file to add:c:\comfile\mainhelp.scr
Name of the procedure file [c:\comfile\compuco1.prc]:
Name of the binary screen block [mainhelp]:

                Help for the COMPUCO Application Main Menu

To print a report about the COMPUCO sales staff performance, select
option 1.

To print a report about the COMPUCO customer base, select option 2.

To quit, select option 3.

                [After you are finished reading this, press any key]

Press any key to continue
```

Screen blocks are not executed directly at the R> prompt but are called using a DISPLAY command from a command block in a procedure file. The command looks like this:

```
DISPLAY scrnname IN filespec
```

*Scrname* is the name given to the binary screen block and *filespec* is the name of the binary procedure file in which the screen resides.

### **Adding an ASCII Menu File to a Procedure File**

Select option 4 from the RCOMPILE main menu to convert an ASCII menu file to a binary menu block and direct the block to a procedure file. The main menu remains on the screen, and RCOMPILE responds with the following message:

Name of the ASCII menu file to add:

Enter the name of the file you want to convert. If the file is not located on the default drive and directory, be sure to include the drive and path specification before the file name. For example, if the file *mainmenu.mnu* is located on drive *c:* in the directory *comfile*, enter:

`c:\comfile\mainmenu.mnu`

RCOMPILE responds with the following message:

Name of the procedure file [c:\comfile\compuco1.prc]:

Press [ENTER] to assign the name shown in brackets to the file, or if no default file exists, enter the name of the new or existing procedure file to which you want to add the converted file. After you press [ENTER], RCOMPILE responds with the following message:

Name of the binary menu block [mainmenu]:

Press [ENTER] to assign the name shown in brackets to the binary block, or assign a new name. After the block name is assigned, RCOMPILE generates the binary file. As each command in the command file is converted to binary form, the ASCII command is displayed on the screen. The screen below shows the three message lines and the result after *mainmenu.mnu* is added. When the addition is complete, you are prompted to press any key on the keyboard to continue.

Name of the ASCII menu file to add:c:\comfile\mainmenu.mnu

Name of the procedure file [c:\comfile\compuco1.prc]:

Name of the binary menu block [mainmenu]:

\$M

mainmenu

column    Select A Report To Print

Display the salesperson report

Display the customer report

Quit

Help

Press any key to continue

Menu blocks are not executed directly at the R> prompt but are called using a CHOOSE command from a command block in a procedure file. The command looks like this:

```
CHOOSE varname FROM menuname IN filespec
```

*Varname* in the variable which holds the operator's selection, *menuname* is the name given to the binary menu block, and *filespec* is the name of the binary procedure file in which the menu resides.

## Converting an ASCII Application File to a Binary Application File

Select option 5 from the RCOMPILE main menu to convert an ASCII application file to a binary application file. The main menu remains on the screen, and RCOMPILE responds with the following message:

Name of the ASCII application file to convert:

Enter the name of the file you want to convert. If the file is not located on the default drive and directory, be sure to include the drive and path specification before the file name. For example, if the file *compu5.app* is located on drive *c:* in the directory *appfile*, enter:

c:\appfile\compu5.app



R\_COMPILE responds with the following message:

```
Name of the procedure file [c:\appfile\compu5.prc]:
```

Press [ENTER] to assign the name shown in brackets to the procedure file. If the procedure file is new, a default name is not displayed and you must assign a name to the file. After the procedure file name is assigned, R\_COMPILE generates the file. If the procedure file exists, the application file will be added to the end as a procedure block. As each command in the application file is converted to binary form, the ASCII command is displayed on the screen. The screen below shows the two message lines and the result after the first ten commands in *compu5.app* are processed. Notice that the first line in the file is *\$COMMAND*. That line indicates that the following lines are a command block. The block ends when another command, menu, or screen block begins. The other two types of blocks, menu and screen, are denoted with *\$MENU* and *\$SCREEN*.

```
Name of the ASCII application file to convert:c:\appfile\compu5.app
```

```
Name of the procedure file [c:\appfile\compu5.asc]:
```

```
$COMMAND
COMPU5
SET MESSAGE OFF
OPEN c:\appfile\janes
SET ERROR MESSAGE OFF
SET VAR pick1 TEXT
SET VAR level1 INT
SET VAR level1 TO 0
WHILE level1 EQ 0 THEN
  NEWPAGE
```

```
Press any key to continue
```

A procedure file is executed with a RUN command. To execute the procedure file properly, you must know the name of the main command block in the procedure file as well as the procedure file name. The command looks like this:

```
RUN blockname IN filespec
```

*Blockname* is the name of the command block and *filespec* is the full name of the binary procedure file including drive and path specifications if the file is not on the current drive and directory.

## Displaying the File Directory

Select option 6 from the RCOMPILE menu to display the file directory. The main menu remains on the screen, and RCOMPILE responds with the following message:

```
Enter DOS drive and subdirectory:
```

Press [ENTER] to display the file directory of the default drive and directory. If the directory you want is not located on the default drive and directory, be sure to include the drive and path specification before the file name. For example, if you want the file directory of the directory *appfile* on drive *c:*, your entry and the RCOMPILE response look similar to the following screen:

```
Enter DOS drive and subdirectory:c: \appfile \
C:COMPUC01.RBS  C:COMPUC02.RBS  C:COMPUC03.RBS  C:MAINMENU.TXT
C:MAINHELP.SCR  C:SALES.CMD      C:CUSTOMER.CMD  C:STARTUP1.CMD
C:MAINMENU.MNU  C:MAINHELP.SCR    C:STARTUP1.API  C:STARTUP1.APX
C:STARTUP1.APP  C:COMPUC01.PRC
```

Note the following filename extension conventions:

- ASCII application files produced by the EXPRESS have the extension *.app*
- ASCII binary files produced by the EXPRESS have the extension *.apx*
- ASCII binary files produced by the EXPRESS for internal use have the extension *.api*
- R:base database files have the extension *.rbs*

Filename extensions for other types of files are up to you. In the preceding directory, the following extension naming conventions are used:

- ASCII application files have the extension *.app*
- ASCII text files used by R:base command files have the extension *.txt*
- Command files have the extension *.cmd*
- Procedure files produced by RCOMPILE have the extension *.prc*
- R:base 5000 screen files have the extension *.scr*
- R:base 5000 menu files have the extension *.mnu*

## Displaying the Contents of an ASCII File

Select option 7 from the RCOMPILE menu to view the contents of an ASCII file. The main menu remains on the screen, and RCOMPILE responds with the following message:

```
Enter the name of the file to be displayed:
```

Enter the file name and, if applicable, drive and path specifications. For example, to view the contents of the file *c:\comfile\mainmenu.mnu*, your entry and the RCOMPILE response look similar to the following screen:

```
Enter the name of the file to be displayed:c:\comfile\mainmenu.mnu
$M
mainmenu
column Select A Report To Print
Display the salesperson report
Display the customer report
Quit
Help

Press any key to continue
```

## Exiting RCOMPILE

Select option 8 to exit RCOMPILE. If you entered RCOMPILE from the R:base 5000 main menu, selecting option 8 returns you to the same menu. If you entered the RCOMPILE command at a DOS prompt, selecting option 8 returns you to an operating system prompt.

---

## Sample Application Contents

<b>How to Use This Chapter</b>	17-2
<b>The Order Application</b>	17-2
<b>Using Order</b>	17-2
The Main Menu	17-3
Order Processing (Option 1)	17-3
Picking, Shipping and Billing (Option 2)	17-3
Customer Data Processing (Option 3)	17-3
Three-up Mailing Labels (Option 4)	17-3
Inventory Data (Option 5)	17-3
System Maintenance (Option 6)	17-3
Getting Started	17-4
Entering Data for a New Customer	17-4
Entering Data for a New Inventory Item	17-5
<b>Order Processing</b>	17-5
Entering a New Order	17-5
Changing/Deleting an Order	17-7
Order Status Reporting	17-7
<b>Picking, Shipping, and Billing</b>	17-7
Entering Goods Shipped	17-8
Billing Customers and Summarizing Sales	17-8
<b>Customer Data Processing</b>	17-9
<b>Printing Mailing Labels</b>	17-10
<b>Inventory Data</b>	17-10
File Maintenance	17-10
Inventory Adjustments	17-11
Inquiry/Report	17-11
<b>System Maintenance</b>	17-11
Database Back Up	17-11
Archive Old Data	17-12
Delete Data	17-13
Maintain System Parameters	17-13
<b>Notes For Programmers</b>	17-14

## HOW TO USE THIS CHAPTER

One of the disks you received with R:base 5000 contains *Order*, an example of a business application developed using the programming capabilities described in chapters 14 and 15. This chapter provides a general description of *Order* together with basic instructions for its use.

## THE ORDER APPLICATION

*Order* was designed for any business that needs to keep track of customers, inventory, and sales orders. When you enter customer sales orders, *Order* automatically updates the inventory. It keeps track of backorders, automatically filling them on a first-come, first-served basis. *Order* prints *picking lists* for use by warehouse personnel. These lists show which items to ship to each customer and the warehouse location of the items. Once the merchandise has been shipped, *Order* prints customer invoices and then automatically provides a sales register for use as a source document for accounting personnel.

In addition to serving as a sales order processing system, *Order* has other useful features. *Order* prints three-up mailing labels either for all customers or only those customers within a range of zip codes which you supply. You can also find out the status of open orders or the status of a particular inventory item.

*Order* prints price lists sorted by item number or description. Accounting personnel can easily obtain a complete listing of all inventory items showing the quantity and cost of inventory on hand and the total cost of the inventory. When it is time to make a physical count of inventory items, *Order* prints a worksheet showing the inventory by warehouse location. *Order* also gives you a list of items you need to order.

To help you become familiar with *Order*, a sample database has been included on your disk.

## USING ORDER

To see how *Order* works, load R:base 5000 following the instructions given in chapter 1. If you have a hard disk system, start from the directory that holds the application files and the sample database. If you have a floppy disk system, place the sample application disk in drive *b:*. At the *R>* prompt, type:

```
RUN init IN procs.ord
```

## The Main Menu

*Order* is menu driven. To make *Order* perform, all you have to do is make choices from the menus which *Order* puts on the screen. The main menu has six options on it. These options tell *Order* which functions you want to work with.

### ORDER PROCESSING (OPTION 1)

Order processing is the part of *Order* where you enter new orders, change unshipped orders, or get information about the status of unshipped orders.

### PICKING, SHIPPING, AND BILLING (OPTION 2)

This part of *Order* is used to print picking lists, record the goods actually shipped, and print invoices and sales registers.

### CUSTOMER DATA PROCESSING (OPTION 3)

Select this option to enter information about a new customer, change existing customer information, delete customer information, print a customer list, or get basic information about a customer.

### THREE-UP MAILING LABELS (OPTION 4)

By making this choice, you can have *Order* print customer mailing labels for all customers or for customers with zip codes within a certain range which you supply.

### INVENTORY DATA (OPTION 5)

This part of *Order* deals with inventory information. You can add, change, and delete information about an inventory item. You can also record adjustments to inventory balances and get information about the status of an item. This is the part of *Order* where you can print various inventory lists including price lists, perpetual inventory list, physical inventory worksheet, and the list of items which you need to order.

### SYSTEM MAINTENANCE (OPTION 6)

There is housekeeping to be done in any system. Here is where it is done for *Order*. With this choice you can make a back-up copy of the *Order* database, delete unwanted data, delete the sample data, and change data which *Order* uses, such as tax rates, next order number, and next invoice number.

Making choices from *Order* menus is easy. For numbered menus like the main menu, just press the number on the keyboard that matches the number of the option you want and then press [ENTER].

## Getting Started

One of the first things you want is a sample price list. To get it, start by selecting option 5, *Inventory Data*, from the main menu. *Order* presents you with another menu. Select option 3, *Inquiry/Report*. *Order* now gives you a list of reports you can get. Select either option 2 or option 3.

*Order* now presents you with another kind of menu. You are asked to indicate where you want the report information to go. You can indicate your option in two ways. The easiest way is to type the first letter of the option you want and then press [ENTER]. For example, if you want the price list printed, press [P] and then press [ENTER]. The other way to select an option from this kind of menu is to move the highlighted area with the cursor control keys until your choice is highlighted, and then press [ENTER].

When *Order* finishes printing the sample price list, it will present you with the *Inventory Data* menu again. Press [ESC], and *Order* will return to the main menu.

Print a sample customer list by following a similar procedure. First select option 3, *Customer Data Processing*, from the main menu. Now select option 5, *Print Customer List*. When you see that menu, send the report to your printer by pressing [P] and [ENTER]. After *Order* is finished printing the list, it will return to the *Customer Data Processing* menu. Return to the main menu by pressing [ESC].

## Entering Data for a New Customer

As you can see, getting information from *Order* is a simple process of selecting menu options. Now try putting data into the database. Record data for a new customer by selecting option 3, *Customer Data Processing*, from the main menu. Select option 1, *Enter A New Customer*.

To record new customer information, simply fill in the CUSTOMER RECORD FORM on the screen. Notice that *Order* has already provided the new customer number. Some of the data is optional, some is required. If you do not want to provide optional data, press [ENTER] to go on to the next item. If you want to change anything before you have finished with the form, you can move from space to space using [ENTER] or the tab key.

When you have finished entering the new customer data, press [ESC]. *Order* then asks if you have more customer data to record. Press [ENTER] to record data for another customer. Press [N] and then press [ENTER] to return to the *Customer Data Processing* menu. Press [ESC] to return to the main menu.

## Entering Data for a New Inventory Item

Entering data for a new inventory item is also an easy procedure. Select option 5, *Inventory Data*, from the main menu. Next select option 1, *File Maintenance*, from the *Inventory Data* menu. Last, select option 1, *Add New Item*, from the *File Maintenance* menu.

*Order* presents you with a blank NEW INVENTORY ITEM FORM. Simply fill in the blanks. All items in this form are required. As with the CUSTOMER RECORD FORM, indicate that you have finished by pressing [ESC]. *Order* returns to the *Inventory Data* menu.

If you wish, you can make a status inquiry of any inventory item by selecting option 3, *Inquiry/Report*, from the *Inventory Data* menu. Now select option 1, *Item Status Inquiry*, from the *Inquiry/Report* menu. *Order* asks you to enter the item number you want to see. You are then given the description, price, unit of measure, number in stock, number backordered, number allocated to orders, and number available for sale for the item you requested. Press any key to return to the *Inventory Data* menu. Then press [ESC] to return to the main menu.

## ORDER PROCESSING

You have learned how to move from the main menu to sub-menus and back again. You have used *Order* to print various reports and make an inventory status inquiry. You have also used *Order* to enter customer and inventory data into the database. Now, you can see how to process a sales order by selecting option 1, *Order Processing*, from the main menu.

## Entering a New Order

Enter a new order by selecting option 1, *Enter A New Order*, from the *Order Processing* menu. *Order* asks you for the customer number of the customer placing the order. This number may come from several sources; the customer may provide it, you can look it up on the customer list you printed, or *Order* can look it up for you.

If you want *Order* to look up a customer number, type [H] and press [ENTER] after *Order* asks you for the customer number. *Order* then asks you to provide a key word or phrase from the company name. For example, if you want to know the customer number for The Great Plains Computer Co, Inc., enter:

Great Plains Com



*Order* then gives you a list of customers with names which contain the phrase *Great Plains Com.* For example, companies with the names Great Plains Communications, Middle Great Plains Compusystems, and Forever Great Plains Company would be listed along with The Great Plains Computer Co. Write down the number you need. When you are ready to proceed, press any key.

Once you have provided *Order* with a valid customer number, *Order* displays the top portion of a sales order form on the screen. Notice that *Order* has filled in the basic customer data. You are asked to provide the customer's purchase order number and the name of the sales representative who made the sale. If you do not wish to provide this data, press [ESC] and go to the next item.

Once the top portion of the order form is complete, *Order* helps you enter information about the merchandise being ordered. For each item to be ordered, *Order* asks you for the item number and the quantity ordered. *Order* fills in the item description, unit of measure, unit price, and total amount.

*Order* tells you the quantity available for shipment. If the quantity entered exceeds the quantity available, *Order* places the excess on a back order waiting list. As additional inventory is received, *Order* fills the backorders on the basis of date and time when the backorders were entered.

Try it out by entering items from the price list you printed. Although there is only space on the screen for about a dozen items, *Order* accepts as many lines as you want. When you are finished entering order lines, press [ENTER] in response to the prompt for an item number.

*Order* looks up the tax and shipping rate for the customer's state, computes the tax and shipping charges, enters them on the order, and calculates the total order amount. At this point *Order* presents a menu which allows you to *Book* the order, *Change* the order or *Delete* the order.

If you choose *Book*, *Order* records the order and returns to the *Order Processing* menu. If you choose *Change*, *Order* displays the order form on the screen and asks you for the item number you want to change. You may change the quantity ordered by entering the new quantity. If you want to delete the item from the order, enter a zero (0) in the *Quantity Ordered* column. If you choose *Delete*, *Order* deletes the entire order, including any backorders, and returns to the *Order Processing* menu.

## Changing/Deleting an Order

After an order has been recorded and before it has been shipped, you may change or cancel it by selecting option 2 from the *Order Processing* menu. *Order* presents a menu which allows you to specify the sales order number (*ORDER#*), the customer's purchase order number (*CUSTOMER PO#*), or *HELP*.

If you select *ORDER#* or *CUSTOMER PO#*, *Order* asks you for the appropriate number and finds the order you want to change. If you select *HELP*, *Order* asks you for a key phrase from the customer's name and shows you the sales order number of all unshipped orders for the customer. You can then write down the number of the order you want to change and provide it to *Order* when you are asked.

## Order Status Reporting

From time to time, you will want to know the status of unshipped orders. *Order* can print an Order Status Report containing unshipped orders for all customers or for any single customer. To obtain an Order Status Report, choose option 3, *Generate Order Status Report*, from the *Order Processing* menu. *Order* presents another menu which allows you to indicate whether you want open orders for all customers or for a given customer. If you want a report for a given customer, *Order* asks you for the customer number. As with all *Order* reports, you are asked to indicate the destination of the report, such as the printer or the screen. When *Order* is finished printing the report, it will return you to the *Order Processing* menu.

## PICKING, SHIPPING, AND BILLING

After orders are recorded, the process of shipping the merchandise ordered is initiated by printing a picking list for the warehouse personnel. To prepare a picking list, choose option 2, *Picking, Shipping And Billing*, from the main menu. Next, choose option 1, *Print Picking List*. *Order* presents a menu from which you can indicate whether you wish to print a list for all unshipped orders or for only a specified order.

The picking list shows which items are to be shipped to each customer. The list is sorted by warehouse location so warehouse personnel can fill the order with the least amount of wasted motion. The list contains a Number-Shipped column. The warehouse personnel write the quantity actually shipped in this column.

When the order is filled, one copy of the picking list can be enclosed with the merchandise to serve as a packing list. The original is returned to serve as a source document for the next step in the process.

## Entering Goods Shipped

Instead of assuming that goods ordered are always shipped, *Order* requires confirmation of shipments. This is done by choosing option 2, *Enter Goods Shipped*, from the *Picking, Shipping And Billing* menu.

*Order* asks you for the sales order number that was shipped. The number can be obtained from the completed picking list. *Order* asks if there are any differences between what was ordered and what was shipped. If there are differences, *Order* asks for the item number and quantity shipped. If there are no differences, press [ENTER] to continue.

When you are finished recording the shipments, *Order* prints an Inventory Exception Report containing those items where the number shipped is different from the number ordered. The Inventory Exception Report is used for control purposes. Sometimes, the quantity of inventory actually on hand is different from the data in the database. The exception report alerts management to a potential problem so that it can be resolved.

## Billing Customers and Summarizing Sales

After the merchandise has been shipped, invoices need to be mailed. Accounting personnel need a summary of the invoices so they can make the accounting entries to record the sales.

These tasks are handled easily by *Order*. By choosing option 3, *Print Invoices*, from the *Picking, Shipping And Billing* menu, *Order* prints invoices for all unbilled Orders which have been shipped. Also, *Order* automatically prints a numbered sales register for the accounting department.

Before printing the invoices, *Order* gives you a chance to prepare the printer. Be certain that the printer is on and that you have loaded the correct paper. *Order* also prints a sample invoice to help you align the printer properly. You may print invoices one at a time or you may use continuous forms. After the invoices are printed, *Order* asks if they were printed properly. If so, press [ENTER]. If not, press [N] and then press [ENTER]. As far as *Order* is concerned, it is as if the invoices had never been printed. This feature is included in case something goes wrong while the invoices are being printed.

You have now been through the entire process of entering sales orders, printing picking lists, recording merchandise shipped, and preparing customer invoices. The remainder of this chapter will deal with other *Order* functions.

## CUSTOMER DATA PROCESSING

To change customer data, choose option 2, *Change Customer Data*, from the *Customer Data Processing* menu (option 3 from the main menu). *Order* asks for the number of the customer whose data you want to change. If you do not have the number, enter [H]. *Order* provides help on the basis of a key word or phrase from the customer name.

Once a valid customer number is entered, *Order* presents a customer record form for the customer you request. The form works the same way as the form you use to record a new customer. You may change any customer information except the customer number. It may never be changed. When you have finished with the changes you want to make, press [ESC]. *Order* records the changes and returns to the *Customer Data Processing* menu.

Customer data may be deleted only if there is no active order data in the database. Basically, only customers who have never placed an order or those for whom all other data has been erased may be deleted. To delete customer data, choose option 3 from the *Customer Data Processing* menu. *Order* asks you for the delete password. The password for the sample database is *sesame*. *Order* then asks you for the customer number to be deleted.

*Order* then checks the database to see if there is any active order data for the customer you entered. If not, *Order* asks if you are sure you want to delete the data. If you change your mind, press [ENTER]. If you want the data deleted, press [Y]. Then press [ENTER]. *Order* returns to the *Customer Data Processing* menu.

To view the information for a customer on the screen, choose option 4, *Customer Inquiry*, from the *Customer Data Processing* menu. *Order* asks for the customer number of the customer whose data you want to see. If you do not have the number, *Order* provides help on the basis of a key word or phrase from the customer's name. *Order* displays the information for the customer you request. When you are ready, press any key. *Order* returns to the *Customer Data Processing* menu.

## PRINTING MAILING LABELS

If you want to print mailing labels for the customers in the database, choose option 4, *Three-up Mailing Labels*, from the main menu. *Order* presents another menu for you to indicate whether you want labels for all the customers in the database or only those customers with ZIP codes within a specific range.

If you want to print only those customers within a specified ZIP code range, *Order* asks you for the lower and upper limits to be included in the range.

*Order* then constructs the data for the mailing labels. For a large database, this process may take a few minutes. When *Order* is ready to print the labels, it will ask you to prepare the printer. Make certain that the printer is turned on and that the mailing labels are set up in the printer. *Order* is designed to work with three-across labels. You may, however, want to test this option with standard paper.

*Order* will print one row of labels to help you align them in the printer. You may repeat this operation as often as needed until you get the labels properly aligned. When *Order* has finished printing the labels, it returns to the *Three-up Mailing Labels* menu.

## INVENTORY DATA

### File Maintenance

To change the data for an inventory item, choose option 2, *Change Item*, from the *File Maintenance* menu. *Order* asks for the item number you want to change. *Order* displays the inventory edit form on the screen with the information for the item you requested. You may change any information in the form except the item number, which may never be changed, and information about quantity on hand, quantity backordered, and quantity allocated to orders. These items may be changed only indirectly with the shipping and inventory adjustment procedures.

An inventory item may be deleted only if there is no order data in the database related to the item and if the quantity on hand is zero. To delete an item which meets these conditions, choose option 3, *Delete Item*, from the *File Maintenance* menu. *Order* asks you for the delete password, which is *sesame*. Next, *Order* asks for the item number of the item you want to delete.

After checking to be sure that the item qualifies for deletion, *Order* displays the information for the item you requested and asks you to confirm that you want the item deleted. If you do not want the item deleted, press [ENTER] and *Order* will ignore the request. If you do want the item deleted, press [Y] and then press [ENTER]. *Order* returns to the *Inventory Data* menu.

## Inventory Adjustments

Record merchandise received by choosing option 2, *Inventory Adjustments*, from the *Inventory Data* menu. *Order* asks for the item number of the item to be adjusted. Once a valid item number is entered, the inventory item adjustment form is displayed. Enter a description of the adjustment, such as "Shipment number 9999 from Midwest Distributors," and the amount of the adjustment. If you are reducing the quantity on hand, type in a minus sign (-) before the amount of the adjustment. When you are finished, press [ESC].

For adjustments which increase the quantity on hand, *Order* automatically fills backorders on a first-come, first-served basis. For adjustments which decrease inventory, *Order* checks to be certain that you are not trying to adjust the quantity on hand to less than zero.

## INQUIRY/REPORT

You have previously printed an inventory price list and seen how to make an item status inquiry. The other reports are obtained in the same way. Try them out. The reports are self explanatory.

## SYSTEM MAINTENANCE

System administration is an important part of any business computer system. Administrative functions of *Order* are explained in this section.

### Database Back Up

Business information, such as that contained in the *Order* database, can be quite valuable. Serious consequences can result if the database is accidentally lost, damaged or destroyed. To minimize the loss if something goes wrong, you need to periodically make a copy of the *Order* database. How often you do this depends on a number of things, including the frequency of changes which are made to the database.

You can easily make a back-up copy of the *Order* database by choosing option 1, *Database Backup*, from the *System Maintenance* menu. *Order* prompts you for the drive or directory where you want the copy written. If you are using a computer with dual floppy drives, press [A] (for drive a:) and press [ENTER].

If your computer has a hard drive:

- Press [A] if you want the copy written to a floppy disk.
- Enter the directory name if you want the copy written to the hard disk. You may not use the same directory that contains *Order*.

If you have indicated that you want the copy written to a floppy disk, *Order* asks you to place a blank, formatted disk in the drive you specified.

Be careful. The *Order* database is in three files designated *order1.rbs*, *order2.rbs* and *order3.rbs*. If the disk or directory you use for back up contains files with these or similar names, the back-up process will overwrite them.

It is a good idea to maintain three copies of the database: the one you are currently using and two back-up copies. Store them in different places. For example, if the original copy of the database is on a hard disk, store one copy in a different subdirectory on the hard disk and another on a floppy disk, or store the back-up copies on separate floppy disks.

Keep your back-up copies up to date. If the data is changed or updated daily, make a new back-up copy at least once a day. To keep all three copies current, make one disk the master and alternate between the other disks to make back-up copies. For example, suppose you have three disks, numbered 1, 2, and 3, that contain identical information. Make disk 1 the master copy and alternate between disks 2 and 3 to make back-up copies. At the end of one day, make a back-up copy of disk 1 on disk 2, and at the end of the following day, make a back-up copy of disk 1 on disk 3.

It does little good to have back-up copies of the database without procedures to ensure that you can recreate the database from the back-up copy. For example, suppose the *Order* database is destroyed at the end of Wednesday before Wednesday's back-up copy is made. You would need the paper copies of the order forms, completed picking lists and inventory change forms for Wednesday's transactions in order to recreate the database from the back-up copy.

## **Archive Old Data**

As time passes, data in the database may become obsolete. Accordingly, you will want to periodically delete unwanted data. To delete old invoice and order data from the *Order* database, choose option 2, *Delete Old Data*, from the *System Maintenance* menu. *Order* asks you for the delete password (*sesame* unless you have changed it). You are then given a series of menus from which you can indicate the data you want deleted.

The first menu asks if you want to delete invoices or orders. Next you are shown the database column names which can be used to qualify the records to be deleted. For example, if you want to delete old data about orders, you can delete order records on the basis of *Order Number*, *Customer Id*, or *Order Date*. The final menu asks what condition you want to use: *equal*, *not equal*, *greater than*, *less than*, *exists*, or *fails*. For example, if you delete all orders before order number 100, you select *Order Number* to qualify the records and *less than* as the condition. When *Order* asks you for the value to use, you enter 100.

*Order* asks if you want the data written to an archive file. It is a good idea to do so. *Order* then asks if you are sure you want to delete the indicated data. If so, type [Y] and press [ENTER]. If not, press [ENTER].

*Order* always prints a copy of the data being deleted. It is a good idea to save this printed copy for a reasonable time. Once the data is deleted, *Order* returns to the main menu.

## Delete Data

You can delete the data by choosing option 3 from the *System Maintenance* menu. After confirming that you want to delete the data, *Order* deletes it and returns to the main menu. This choice always deletes all data from the database.

## Maintain System Parameters

*Order* uses certain items of information which are provided by the system during processing. These items are listed in table 17-1.

Table 17-1 Information Items for *Order*

Item Name	Item Description
Company name	Company name of customer. <i>Order</i> uses this data to print various report headings and so forth.
Next order number	Next order number which <i>Order</i> assigns.
Next invoice number	Next invoice number to be assigned.
Next customer number	Next customer number to be assigned.
Inventory cost method	Cost method is used for inventory, such as LIFO, FIFO, and AVERAGE.
Delete password	Password needed to delete data from the database.
Back-up date	Date of the database's last back up.
Back-up time	Time of the database's last back up.

You must have the delete password (originally *sesame*) in order to change any of the system parameters. They are changed by choosing option 4, *Maintain System Parameters*, from the *System Maintenance* menu. After providing the delete password, *Order* presents you with a form containing the system parameters. When you have finished with your changes, press [ESC] to return to the main menu.



## NOTES TO PROGRAMMERS

The command procedures used in *Order* are included on the disk. They have been provided to illustrate the use of the R:base 5000 programming language.

There are three basic types of *Order* procedures: menu procedures, processing procedures, and macro procedures. A menu procedure displays the menu, uses the CHOOSE command to get the user's choice, and passes control to another procedure. A processing procedure performs a given *Order* task such as order entry. A macro procedure is a procedure which is used in many different places in *Order*. An example is the GETCUS procedure, which is used whenever a customer number is required from the user.

To get an idea of how *Order* is structured, browse through the procedures using the R:base or DOS TYPE command. Begin with the *procs2.asc* file. You will see that the command block INIT passes control to the main menu processing file, *menus.ord*. The main menu processing file then passes control to the six sub-menu processors, *men1proc*, *men2proc*, and so forth. You can follow the entire application in this way.

You may use any of the command procedures to build applications of your own. It is best to copy the procedure you want to change to another file before modifying it. That way you will always have the original version available to you.

**INDEX WITH GLOSSARY****A**

- Abbreviations, command ..... 5-9
- Adding
  - ASCII command files to procedure files ..... 16-7
  - ASCII menu files to procedure files ..... 16-9
  - ASCII screen files to procedure files ..... 16-8
  - New tables and columns ..... 6-15,14-13
  - Other Microrim products to the R:base 5000 main menu ..... 1-13
- Adjusting the screen display width ..... 5-30
- Advanced report concepts ..... 11-48
- Alphanumeric
  - Consisting of both alphabetic and numeric symbols.
- APPEND ..... 10-5
  - R:base command. Adds rows from one table to another table.
- Application ..... 4-2,14-18,17-2
  - A specific program or task to which a computer solution may be applied.
- Application
  - Application files ..... 14-20
  - Command files ..... 14-19
  - Form and report files ..... 14-20
  - Menu files ..... 14-19
  - Operator ..... 2-24,2-25
  - Owner ..... 2-24,2-26
  - Screen files ..... 14-19
- Application definition. .... 4-5,4-10,4-12,4-13,4-16,14-18
- Application design with the EXPRESS ..... 2-27,4-2,14-18

**Application EXPRESS**

The program used to build R:base databases and applications.

Actions .....	4-6,14-24
Browse .....	14-26
Custom .....	14-27
Delete .....	14-26
Edit .....	14-25
Exit .....	14-27
Load .....	14-24
Macro.....	14-27
Menu.....	4-6,14-27
Print .....	14-26
Select .....	14-26
Application redefinition .....	14-28
Application files .....	14-20
Database definition .....	3-6,14-10
Database redefinition .....	14-13
Help mode .....	14-5
Keyboard functions .....	14-8
Main menu.....	3-6,4-4,14-3
Screen format .....	14-5
Using the EXPRESS .....	3-6,4-2,14-5
Developer .....	2-24,2-26
Operator.....	2-24
Owner .....	2-24
Arguments.....	5-7
A list of conditions or requirements used with a command to pass information or to define precisely what the command is to do.	
Arithmetic calculations .....	8-8
Arithmetic operators .....	5-15,11-15
Signs used in expressions to indicate the type of operation to perform. For example, +, -, ×, /, >, <.	
Array	
A data structure organizing information into specified groups accessed by an index entry. (See Vector).	
Ascending order .....	5-13
Sorting in an order from lowest to highest (such as 1 to 50, A to Z). Sorting follows the order of ASCII character codes.	
Sorting by .....	5-13
AS ASCII clause .....	12-4
AS DIF clause .....	12-4

AS MPLAN clause .....	12-4
ASCII delimiter character .....	12-12,13-39
ASCII delimited files .....	12-12,13-39
ASCII file which divides individual fields in a record with a marking character such as a slash ( / ) or comma ( , ).	
Delimiter character .....	12-12,13-39
File conversion .....	13-39
LOAD/UNLOAD ... AS ASCII .....	12-5,12-12,12-15
What you need to know before conversion .....	13-39
ASCII files .....	12-10,13-35,13-30
American Standard Code for Information Interchange standard file format.	
ASCII is used as a standard format in communication between computers and computer devices.	
Delimited format, transfer of .....	12-12,13-39
Fixed field format, transfer of .....	12-15,13-35
ASCII fixed field files .....	12-15,13-35
ASCII file which determines field boundaries by assigning a specific length to each field.	
File conversion .....	13-35
Input screen, FileGateway .....	13-36
What you need to know before conversion .....	13-35
ASSIGN .....	9-2
R:base command. Allows you to change the value of a column using an expression.	
Command expressions .....	9-3
Command operators .....	9-3
Assigning calculated values to columns .....	9-2
Assigning passwords .....	6-9
AT scrnrow scrncol .....	15-23,15-24
Specifies screen coordinates for various R:base commands such as WRITE, FILLIN.	
Attribute (See column.)	
AUTOSKIP .....	5-27
R:base SET command keyword. When on, automatically skips to next data entry field when the current field is filled.	
AVE .....	8-9
R:base COMPUTE command modifier. Average.	

**B**

Backing up disks..... 1-7,5-33,17-11

BELL..... 5-27

R:base SET command keyword. Sets the terminal speaker off or on.

**Bit**

Binary digit. The basic unit of information in a computer system. Each bit has two possible values—0 or 1. It is the smallest unit recognized by a computer. Eight bits form a byte.

BLANK..... 5-26

R:base SET command keyword. Allows blank character to be defined as any desired character. For example, BLANK=#.

**Boolean**

Pertains to a logical combination system which compares values through the operators LT, LE, GT, GE, EQ, and NE, and connects comparisons with the operators AND and OR.

BREAK.....15-18

R:base command. Used in command files with a WHILE...ENDWHILE loop to enable early exit from the loop.

Breakpoints.....11-40

In reports, indicates text to be printed when a specified column value changes.

Browse.....14-18

Buffers..... 1-16

Building applications with EXPRESS..... 4-2,14-18

Building custom macros..... 4-24,14-27,15-3

BUILD KEY..... 6-17

R:base command. Builds a data location index entry in one table for a column.

**Byte**

Eight bits. Words and numbers are represented by combinations of bytes. The storage capacity of computers is represented in bytes, such as 256K bytes.

**C****Calculating**

Column totals..... 3-31,8-8,11-16

Percentages.....11-17

Report subtotals.....11-16

Report totals.....11-17

Variables in reports..... 3-31,11-14

- 
- CASE ..... 5-30  
 R:base SET command keyword. If CASE is on, R:base differentiates between upper and lower case letters. If off, case is ignored.
- CD ..... 5-37  
 Abbreviation for CHDIR (Change Directory).
- CHANGE ..... 9-5  
 R:base command. A data modification command that changes the value of a column.
- CHANGE COLUMN ..... 6-16  
 R:base command. Used to redefine a column name, type, length or any one or more of these values.
- Changing
- Column and variable locations in reports ..... 3-2,11-24
  - Column values ..... 9-5
  - PROMPT.DAT file ..... 15-28
  - R:base prompts ..... 15-27
  - Special characters ..... 5-26
  - Variable data type ..... 15-10
- Changing an existing application with the EXPRESS ..... 14-28
- Adding menu options ..... 14-29
  - Changing existing actions ..... 14-32
  - Changing menu text and options ..... 14-29
  - Changing menu titles ..... 14-29
  - Changing the escape action ..... 14-30
  - Changing the help action ..... 14-31
  - Changing the name ..... 14-29
  - Deleting menu options ..... 14-30
  - Inserting menu options ..... 14-30
  - Removing menus ..... 14-33
- Changing an existing database definition
- Adding columns to tables ..... 6-17,14-15
  - Adding tables ..... 6-8,14-14
  - Changing column names, data types, and lengths ..... 6-16,14-15
  - Changing table names ..... 6-20,14-14
  - Creating and deleting column keys ..... 6-17
  - Deleting rules ..... 6-20
  - Owner passwords ..... 6-19
  - Removing columns from tables ..... 6-18,14-17
  - Removing tables ..... 6-18,14-17

**Character**

A single printable letter (A-Z), numeral (0-9), or symbol (such as #, \$, %) used to represent data. Includes invisible characters such as space, tab, and carriage return.

**CHDIR** ..... 5-37

R:base command. Used to change the current directory.

**CHDRV** ..... 5-37

R:base command. Used to change the default drive.

**CHECK** ..... 7-20

R:base LOAD command modifier. Same function as the SET RULES ON command when in load mode. Enables rule checking while loading.

**CHKDSK** ..... 5-37

R:base command. Used to display total and remaining disk and memory capacity.

**CHOOSE** ..... 15-25

R:base command. Used in a command file to display a stored menu and enter an operator response into an R:base variable.

Choosing a report name ..... 3-29,11-8

chrname ..... 5-7,5-26

A settable definition for special characters including: BLANK, DOLLAR, QUOTES, SEMI, PLUS, and COMMA.

**CLEAR** ..... 5-27,15-8

R:base command. Used to clear variable values from memory. Also a SET command keyword used to indicate the buffer clearing status, on or off.

**CLOSE** ..... 5-22

R:base command. Used to properly release a database from open status.

cmdfile ..... 5-7

A file containing R:base commands. It has the same format as filespec.

collist ..... 5-7

A list of columns used in the command syntax diagrams.

colname ..... 5-7

The name of a column used in the command syntax diagrams.

**COLOR** ..... 1-12,1-14,5-27

R:base SET command keyword. Used with FOREGRND or BACKGRND to set the foreground and background screen colors for the R:base 5000 main menu and R:base.

**Column** ..... 6-6

A specific piece of information defined for a database table. In some products, called a field or attribute.

Description ..... 2-2, 2-5

- 
- Column cursor  
Reverse video marker used to highlight a column value during editing, entering, etc.
- COLUMNS ..... 6-6  
R:base DEFINE mode command. Defines the name, data type, key designation, and, if TEXT, the length of each column.
- Columns, reports ..... 11-24  
    Changing locations ..... 11-29  
    Deleting locations ..... 11-29
- Column totals ..... 8-8,11-16
- Combining tables ..... 10-2
- Combining TEXT expressions ..... 11-21,15-12
- Combining two TEXT columns ..... 11-21
- COMMA ..... 5-26  
R:base SET command keyword. Allows comma character to be defined as any desired character. For example, COMMA=\*
- \$COMMAND ..... 16-2  
Definition line used in a procedure file to indicate the following code is a command block.
- Command ..... 5-9  
A user instruction to the computer, generally entered on a keyboard. This can be a word, abbreviation or character that directs the system to perform a predefined operation.  
    Abbreviations ..... 5-9  
    Command comments ..... 5-9  
    Command summary ..... 5-5  
    Command syntax arguments ..... 5-7  
    Command syntax keywords ..... 5-6  
    Continuation characters ..... 5-10  
    Multiple commands on a line ..... 5-10  
    Syntax ..... 5-4
- Command file ..... 15-3  
A file containing a series of R:base commands that R:base executes when the file is invoked using the INPUT or RUN commands.  
    Debugging ..... 15-48  
    Editing ..... 15-4  
    Executing ..... 15-7  
    Initial command file ..... 1-14,14-28,15-51  
    Saving ..... 14-27,15-6
- \*(Comments) ..... 5-9,15-8



Compiler (see RCOMPILE.)	16-2
COMPUTE	8-8
R:base command. Computes simple functional values for the selected column and optionally allows you to store the value in a global variable.	
Concatenating TEXT columns and values	11-21,15-12
Condition	5-14,15-13
A specification that is either TRUE or FALSE. Conditions are used in WHERE clauses and in the IF and WHILE commands.	
Conditional commands: IF...ENDIF	15-13
Conditional commands: WHILE...ENDW	15-17
condlist	5-7,15-13,15-16,15-17
A list of conditions that identifies the rows to be referenced. Used with WHERE, IF...THEN, and WHILE...ENDWHILE.	
CONTAINS	15-16
R:base WHERE clause modifier. Used to compare the values of TEXT strings.	
Continuation of commands	5-10
CONVERT	1-16
R:base module used to convert R:base 4000 files to R:base 5000 formats.	
Converting	
ASCII delimited file to R:base format	12-12,13-39
ASCII fixed field file to R:base format	12-15,13-35
ASCII command files to binary command files	16-5
ASCII application files to binary application files	16-11
dBASE II file to R:base format	13-46
DIF file to R:base format	13-27
Lotus worksheet to R:base format	13-41
Multiplan SYLK file to R:base format	13-30
PFS:FILE to R:base format	13-51
R:base table to ASCII delimited format	12-9
R:base table to DIF format	12-7
R:base table to Multiplan SYLK format	12-8
R:base table to another database	12-18
SYLK file to R:base format	13-30
VisiCalc file to DIF format	13-28
COPY	5-37
R:base command. Used to copy external files.	
Copying reports to a different database	11-54

COUNT ..... 8-9  
 R:base COMPUTE command modifier. Used to calculate the total number of rows in a specific column.

#### Creating

Columns ..... 3-8,6-6,14-12  
 Column keys ..... 6-7,6-17  
 Command files ..... 14-19,15-3  
 Custom macros ..... 4-17,14-27,15-49  
 Databases ..... 3-6,6-2,14-10  
 Forms ..... 7-2  
 table forms ..... 7-3,14-24  
 variable forms ..... 7-3,15-32  
 Menus in R:base ..... 4-5,4-10,4-12,4-13,4-16,14-19,15-23  
 Reports ..... 3-28,11-7,14-26  
 Tables ..... 3-7,6-8,14-11  
 from existing tables ..... 10-5  
 Variables ..... 11-4,15-8

#### CRT

Cathode Ray Tube. This is the screen on a computer terminal.

#### Cursor

A movable marker or line on the terminal screen that designates the next point of character entry or change. (See column cursor.)

#### Customized command files

Data entry forms ..... 7-2  
 Prompt displays ..... 15-27  
 R:base 5000 main menu ..... 1-11

## D

#### Data

Facts, numbers, letters, and symbols stored in the computer. Data can be thought of as the basic elements of information used, created, or otherwise processed by an application program.

Entry ..... 7-2  
 Types ..... 6-6  
 Verification ..... 6-11

**Database**

A collection of logical groups of data stored in one or more files that are used to perform a task.

Access from a command file .....	15-3
Backup and recovery .....	5-33
Design .....	2-3
Definition .....	3-6,6-2,14-10
Description .....	2-2
Loading .....	2-20,7-14
Model .....	2-3
Modification .....	6-15,14-13
Passwords .....	6-9
Query .....	8-2
Structure .....	3-17

Database definition options .....	6-2
-----------------------------------	-----

**Database design**

Implementation .....	2-18
Improvement .....	2-8
Steps .....	2-6

Database load menu, FileGateway .....	13-20
---------------------------------------	-------

**Database Management System (DBMS)**

A means for storing information in a systematic order. Libraries, dictionaries, and file cabinets are all examples of database management systems. Commonly called DBMS.

Description .....	2-2
-------------------	-----

Database name .....	3-7,6-5,14-11
---------------------	---------------

The database name is in the form *d:\path\dbname* where *d*: is the drive designation, *path* is the directory path, and *dbname* is seven characters or less database name. No extension is allowed. R:base assigns the extension *.RBS* to the database when it is stored. The three DOS files called *dbname1.RBS*, *dbname2.RBS*, and *dbname3.RBS*.

Database relationship links .....	2-15,2-17
-----------------------------------	-----------

**Database schema (See Schema.)****Database terms**

Attributes .....	13-7
Columns .....	2-2,2-5,13-7
Database .....	2-2,13-7
Fields .....	13-7
Files .....	13-7
Records .....	13-7
Relations .....	13-7
Rows .....	2-5,13-7
Tables .....	2-2,2-5,13-7

Data entry methods .....	7-2
Data entry RULES .....	6-11
Data model design .....	2-3
Data types	
Recognized formats in which data may be represented to R:base.	
DATE .....	6-6
DOLLAR .....	6-6
INTEGER .....	6-6
REAL .....	6-6
TEXT .....	6-6
TIME .....	6-6
VARIABLE .....	15-9
#DATE .....	11-20,15-13
DATE .....	5-28,6-6,14-12,15-10
R:base SET command. Keyword used to reset the system date format.	
FileGateway format .....	13-8
Date Arithmetic .....	9-4
Date format selection menu, FileGateway .....	13-21
dBASE II	
Ashton-Tate's Database Management System. The files may be converted by the FileGateway for use with R:base.	
Data formats .....	13-46
Delimiter .....	12-12
File conversion, FileGateway .....	13-46
dBASE III .....	12-12,13-46
Ashton-Tate's Database Management System.	
dbname .....	5-7
A database name.	
dbspec .....	5-7
A one to seven character database name plus the drive and directory if the database is not on the current directory. In the form, <i>d:\path\dbname</i> where <i>d</i> : is the optional drive letter, <i>path</i> is the optional directory path, and <i>dbname</i> is the 1-7 character database name.	
Debugging command files .....	15-48
DEFINE .....	6-5
R:base command. Enters the define mode for a new or existing database.	
Enables definition of all database characteristics.	

**Defining**

- A database ..... 3-6,6-2,14-10
  - A form ..... 3-21,7-2,14-24,15-32
  - A report ..... 3-28,11-7,14-26
  - Application menu options ..... 4-5,14-22
  - Application reports ..... 4-13,4-20,14-26
  - Columns ..... 3-8,6-6,14-12
  - Data types ..... 3-8,14-12
  - Report variables ..... 3-31,11-14
  - Rules ..... 6-11
  - Table passwords ..... 6-9
  - Tables ..... 3-7,6-8,14-11
  - Variables ..... 11-14,15-8
- Defining a database with the Application EXPRESS ..... 3-6,14-10
- Naming a database ..... 3-7,14-11
  - Naming a column ..... 3-8,14-12
  - Naming a table ..... 3-7,14-11
  - Selecting column data types ..... 38,14-12
- Defining an application with the EXPRESS ..... 4-2,14-18
- Selecting a database ..... 4-4,14-18
  - Naming the application ..... 4-4,14-21
  - Naming the main menu ..... 4-4,14-21
  - Selecting menu types ..... 4-4,14-21
  - Designing help screens for the main menu ..... 14-22
  - Entering the main menu title ..... 14-21
  - Entering text for menu options ..... 4-5,14-22
  - Selecting actions for main menu options ..... 4-6,14-23
- Defining, deleting, and changing report variables ..... 11-14
- DELETE ..... 9-7
- R:base command. Used to operating system files.
- DELETE DUPLICATES ..... 9-8
- R:base command. Removes duplicate rows from a table.
- DELETE KEY ..... 6-17
- R:base command. Removes the Key designation from a specified column.
- DELETE ROWS ..... 9-7
- R:base command. Removes specified rows from a table.

Deleting	
Column keys	6-17
Columns from tables	6-18
Forms	7-13
Reports	11-48
Rows from tables	9-7
Rules	6-20
Tables from databases	6-18
Variables	11-23,15-8
Delimiter	
A character used to separate fields in a data record.	
ASCII delimited format	12-5,12-12,13-35,13-39
SET command	5-28
Descending order	5-13
Sorting in an order from highest to lowest (such as 50 to 1, Z to A). Sorting follows the order of ASCII character codes.	
Designing a database	2-3
Designing menus	2-28
Detail line	11-38
On a report, a line used to print columns and variables.	
DIF Files	
A file format used by such products as VisiCalc to transfer data from one system to another.	
File conversion, FileGateway	13-29
UNLOAD ... AS DIF	12-7
DIR	5-37
R:base command. Lists the files on the specified drive or drive and directory.	
Directory	
A logical subdivision of the disk space containing files, usually used to organize files into desired groups.	
Disk	
A magnetic storage device on which data is stored in circular tracks on the surface. The data can be read by the computer's disk drive.	
Disk Contents, R:base 5000	1-19
DISPLAY	15-25
R:base command. Used to display a screen stored in a command file.	



**Displaying**

Application structure .....	4-12,14-6
Columns .....	3-18,6-24
Contents of ASCII files .....	16-14
Data .....	3-14,8-2
Database names .....	6-21
Database structure .....	3-17,6-20
File directories .....	5-37,14-34,16-13
Form names .....	6-25
Forms layout .....	7-12
Report names .....	6-25
Rules .....	6-24
System default conditions .....	5-32
Table names .....	3-17,6-21
Table structure .....	3-17,6-22
Variables .....	3-31,15-48
Width default values .....	5-32

**DOLLAR** ..... 6-6,14-12,15-10

A data type representing dollar amounts in the range of 0 to +  
or -\$99,999,999,999.999. If no decimal point is included, .00 is assumed.  
Also, a SET command keyword used to indicate the special character used in a  
dollar format, usually \$.

**Dollar mask for reports** .....11-29**Dollar special character** ..... 5-26**DOS**

Disk Operating System. Short form of PC-DOS.

**DRAW** .....15-35

R:base command. Used to display a variables form.

**Drive**

The computer component used to read and write data on magnetic storage  
devices such as disks.

**Drive specification or designation**

A letter assigned to a logical division of a disk drive system. For example,  
drive *a*:

**Duplicating reports within a database** .....11-53

**E**

- ECHO** ..... 5-28  
 R:base SET command keyword. If on, causes all commands and data from the current input device to be listed on the current output device.
- EDIT** ..... 9-9  
 R:base command. Used to browse through selected rows of a table, changing, adding, updating, or deleting rows.
- EDIT USING** ..... 9-14  
 R:base command. Used to change, and update, or delete rows using the display format of the specified table form.
- EDIT VARIABLE** ..... 15-37  
 R:base command. Used to edit the value of an applications variable, usually in conjunction with a variable form.
- Editing ASCII fixed field files, FileGateway ..... 13-37, 13-40
- Editing
- Commands ..... 5-11
  - Command files ..... 15-4
  - Data ..... 9-9
    - with forms ..... 9-13
    - with standard display ..... 9-9
  - Exceptions, FileGateway ..... 13-23
  - Forms ..... 7-5
  - Reports ..... 11-11
  - Variables ..... 15-37
- Editor keyboard functions, reports ..... 11-12
- Editor, R:base (RBEDIT) ..... 15-4
- END**  
 R:base command. Used to terminate DEFINE, LOAD, PROMPT, or HELP modes.
- Define mode ..... 6-15
  - Help mode ..... 5-20
  - Load mode ..... 7-16
  - Prompt mode ..... 5-22
- ENTER**  
 R:base command. Enters data into a table.
- With a table form ..... 7-14
  - With a variable form ..... 7-16, 15-36
- ENTER VARIABLE** ..... 15-36  
 R:base command. Used to enter the value of an applications variable, usually in conjunction with a variable form.



Entering Commands (See Command.)

ERASE ..... 5-37  
R:base command. Used to delete operating system files.

Error Message

Text displayed by the computer when an entry is incorrect or some other condition exists where the system is unable to respond to a command. Error messages are listed in the *R:base 5000 Reference Manual*.

Error variables ..... 5-31,15-46

ESCAPE ..... 5-29  
R:base SET command keyword. Used to enable or disable the [ESC] key during command file processing.

Establishing variable data types ..... 15-9

Estimating file sizes ..... 5-38

Exception handling editor, FileGateway ..... 13-23

Exception handling options menu, FileGateway ..... 13-25

Exception loading menu, FileGateway ..... 13-25

Existing table name menu, FileGateway ..... 13-17

EXISTS ..... 5-16  
R:base WHERE clause modifier. A column value must be present to satisfy this condition.

EXIT ..... 3-12,3-41,5-23  
R:base command. Leaves R:base and returns to the operating system.

Exiting the EXPRESS ..... 14-34

Exiting RCOMPILE ..... 16-14

EXPAND ..... 6-17  
R:base command. Used to add new or existing columns to an existing table.

EXPRESS (See Application EXPRESS.)

Expression

An arithmetic formula or value.

Defining ..... 15-11

Evaluating ..... 15-11

Guidelines for ..... 11-14,15-8

Specific values or arithmetic equations ..... 15-11

Used with ASSIGN ..... 9-4

Used with SET VARIABLE ..... 15-11

Used with REPORTS ..... 11-14

**F**

- FAILS** ..... 5-16  
 R:base WHERE clause modifier. This condition is satisfied when the column value is null, usually -0-.
- Features and capabilities** ..... 1-18
- Field (See Column.)**
- Files (DOS)** ..... 3-11,5-37  
 Data that is stored on a disk and that is given a unique file name. An R:base database is stored in three DOS files: *dbname1.RBS*, *dbname2.RBS*, and *dbname3.RBS* (where *dbname* is the name of the database).  
     Estimating sizes of database files ..... 5-38  
     R:base 5000 program files ..... 1-6,1-9  
     Structure of R:base files ..... 5-37
- File conversion editor, FileGateway** ..... 13-19
- File extensions, FileGateway** ..... 13-7
- File type selection menu, FileGateway** ..... 13-15
- FileGateway**  
 The R:base file conversion processor. This utility transfers VisiCalc, Lotus 1-2-3, Multiplan, dBASE II, PFS:FILE, and ASCII files into an R:base database format. (See the individual products.)  
     Database load menu ..... 13-20  
     Date format selection menu ..... 13-21  
     Exception handling editor ..... 13-23  
     Exception handling options ..... 13-25  
     Exception loading menu ..... 13-24  
     Existing table name menu ..... 13-17  
     File conversion editor ..... 13-19  
     Help options menu ..... 13-13  
     Leaving the program ..... 13-26
- filename** ..... 5-7  
 A one to eight character file name. Only used in reference to DOS files. R:base database data groups are known as tables.
- filespec** ..... 5-7  
 Complete specification of a file. In the form *d:\path\filename.ext* where *d*: is the drive on which the file resides, *path* is the directory specification, *filename* is the one to eight character name of the file, and *.ext* is the one to three character file extension.

- FILL** ..... 7-18  
R:base LOAD command modifier. Allows entry of incomplete rows with the LOAD command. Normally, all columns must be entered for each row in entry mode.
- FILLIN** ..... 15-24,15-25  
R:base command. Prompts for the value of a global variable during entry. Only used in command files.
- Footing lines in reports** ..... 3-33,11-30,11-32,11-35
- formname** ..... 5-7  
The name given a form when invoking the FORMS processor.
- FORMS** ..... 7-2  
R:base command. Enters FORMS mode allowing you to create a new form or modify an existing form. Also the table which holds user-generated forms.  
    Table forms mode ..... 7-3  
    Variable forms mode ..... 7-3  
    Locating columns ..... 7-7  
    Locating variables ..... 15-22
- Forms maintenance** ..... 7-12
- Function keys** ..... 3-23,3-30,7-6,11-13,14-8,15-5  
Keys used for display and editing functions.

## G

- GATEWAY** ..... 1-9,1-19,13-14  
Command used at the DOS level or from the R:base 5000 main menu to execute the FileGateway subsystem.
- Generating a customized prompt file** ..... 15-27
- Global variable** ..... 5-31,15-8  
A temporary (not saved) variable that is used in command files, conditional command processing, and to transfer information between tables.
- GOTO** ..... 15-20  
R:base command. Used in a command file to transfer control to a label defined by the LABEL command in the command file.

## H

- Heading lines in reports** ..... 3-33,11-30,11-32,11-35

- 
- HELP ..... 3-4,5-18  
 R:base command. Provides an on-line command list with syntax and description.
- Help menus, FileGateway ..... 13-13
- I**
- IF ... THEN ... ELSE ... ENDIF ..... 15-15  
 R:base command. Used in a command file to execute a block of commands under certain conditions and, if the conditions are not met, to execute an alternate block of commands following the optional ELSE.
- Implementing menus in R:base ..... 15-23
- Indenting text wrap in reports ..... 11-26
- Index (See KEY.)
- Initial command file ..... 15-51  
 A command file that executes automatically when you load R:base. See RBASE.DAT.
- Initial command files, uses of ..... 15-51
- INPUT ..... 15-7  
 R:base command. Changes the input source from the keyboard to a file or back to the keyboard.
- Input device  
 A device, such as the keyboard or a disk drive, attached to the computer from which data and commands can be read. See Output device.
- Input file ..... 12-10  
 A file that holds data and/or commands which may be read and acted upon by R:base or the FileGateway. (See Output file.)  
 Editing ..... 12-20,13-19
- Input/Output commands in command files ..... 15-22
- Input sources, selection of ..... 15-23
- Installing custom macros ..... 4-25,14-27
- Installing R:base 5000 on a hard disk ..... 1-3
- Installing R:base with a floppy disk system ..... 1-7
- Instruction  
 A command that tells the computer what operation to perform.
- INTEGER ..... 6-6,14-12,15-10  
 Data type representing whole numbers (no fractional or decimal parts) in the range of 0 to + or -999,999,999.

Integrating command files .....15-48

**Interactive**

Two way communication between user and computer (or software). For example, the user issues a command and R:base responds immediately.

Interactive EDIT command (See EDIT.)

INTERSECT ..... 2-23,3-20,10-9

R:base command. Used to form a new table from two existing tables using rows with common matching values.

**J**

JOIN .....10-13

R:base command. A relational command used to form a new table from two existing tables. All columns in both source tables are combined when a specified comparison between a column in each table is met.

**K**

**Key**

An index entry for a column which allows R:base to locate and process the column more quickly.

Keyed access for better performance ..... 6-7,10-2,10-14

**KEYBOARD**

Used with the INPUT command to specify the keyboard as the input device.

**Keyboard**

The typewriter-like component of a computer that allows you to enter information.

Key columns ..... 6-7

keylist .....15-13,15-36,15-37

One or more special key designators used to indicate when to exit from an application and when to page up or page down. These keys are specified as ESC, ENTER, PGUP, and PGDN.

Key processing with the WHERE clause ..... 5-17,10-2,10-14

Keywords .....5-6

R:base functions used to determine the R:base environment. These include AUTOSKIP, BELL, CASE, CLEAR, COLOR, DATE, ECHO, ESCAPE, LINES, MESSAGE, NULL, REVERSE, RULES, SCRATCH, USER, and WIDTH. (See SET and SHOW.)

**L**

- LABEL** ..... 15-20  
 R:base command. Defines a label for the GOTO command.
- Labels, mailing** ..... 11-49, 17-3, 17-10
- Leaving R:base** ..... 3-41
- Length** ..... 6-6  
 The number of characters a text column or global variable may contain.  
 Maximum length for TEXT items is 1500 characters.
- LIMIT** ..... 5-16  
 R:base WHERE clause modifier. Used to limit the number of rows retrieved.
- LINES** ..... 5-29  
 R:base SET command keyword. Sets the number of lines of data to be sent to the current output device before a page eject occurs. Does not affect report generation.
- Lines per page on reports** ..... 11-44
- LIST ALL** ..... 6-22  
 R:base command. Lists all tables and their structures in the open database.
- LIST COLUMNS** ..... 6-24  
 R:base command. Lists all columns and their definitions in the open database.
- LIST DATABASES** ..... 6-21  
 R:base command. Lists all databases available on the current directory.
- LIST FORMS** ..... 6-25  
 R:base command. Lists user-defined forms in the FORMS table.
- LIST REPORTS** ..... 6-25  
 R:base command. Lists user-defined reports in the REPORTS table.
- LIST RULES** ..... 6-24  
 R:base command. Lists user-defined rules in the RULES table.
- LIST TABLES** ..... 6-21  
 R:base command. Lists tables if no argument, or full table definition of a specified table.
- LOAD** ..... 12-10  
 R:base command. Adds data to a specified table from an external file or from the keyboard if no file is specified.  
     ... AS ASCII ..... 12-12  
     ... WITH PROMPTS ..... 7-16, 3-13  
     ... WITHOUT PROMPTS ..... 7-17

**Loading**

ASCII Delimited files into R:base .....	12-13,13-39
ASCII Fixed field files into R:base .....	12-15,13-35
Data from a file .....	12-10
Data with null values .....	12-14
Data with prompts .....	3-13,7-16
Data without prompts .....	7-17
dBASE II files into R:base .....	13-46
DIF files into R:base .....	13-27
Lotus files into R:base .....	13-41
PFS:FILE into R:base .....	13-51
SYLK files into R:base .....	13-30

Locating columns and variables on a report .....	3-32,11-25
--	------------

Locating columns on a form .....	7-7
----------------------------------	-----

Location selection menu, Reports .....	11-24
--	-------

Long entries .....	5-10
--------------------	------

Lookup .....	3-31,11-18
--------------	------------

A value taken from a table for use in a Report

**Lotus 1-2-3**

A Lotus Development Corporation spreadsheet application system.

Data formats .....	13-41
File conversion procedures .....	13-43
Spreadsheets .....	13-9
What you need to know before conversion .....	13-41

**M**

Macros .....	4-17,14-27,15-50
Sets of instructions grouped together in an efficient way, designed for speed and convenience in streamlining repetitive tasks. A command file.	

Mailing labels .....	11-49,17-3,17-10
----------------------	------------------

MailMerge .....	12-22
A Micropro product. Addition to the WordStar word processing system.	

Mainframe RIM .....	12-6
Relational Information Management for large-scale computers. RIM was originally designed to work on mainframe computers. R:base evolved on microcomputers from the same original program.	

Main menu, R:base .....	1-6,1-11,3-5
-------------------------	--------------

Maintaining a database .....	3-37
------------------------------	------

Maintaining forms .....	7-12
-------------------------	------

- 
- Managing databases and applications that are on the same disk ..... 15-52
- Many-to-many database relationships ..... 2-17
- Marking report lines ..... 3-33,11-30
- Breakpoints ..... 11-40
  - Detail ..... 11-38
  - Page Footing ..... 11-35
  - Page Heading ..... 11-35
  - Report Footing ..... 3-34,11-32
  - Report Heading ..... 3-33,11-32
  - Subheadings ..... 11-40
  - Subfootings ..... 11-40
- MAX ..... 8-9
- R:base COMPUTE command modifier. Used to determine the maximum row value for a specific column.
- MD ..... 5-37
- Abbreviation for MKDIR (Make Directory.)
- Memory
- Temporary storage area in a computer which holds a program and data when a program is run. RAM, Random Access Memory.
- \$MENU ..... 16-2
- Definition line used in a procedure file to indicate the following code is a menu block.
- Menu ..... 4-4,11-3,13-15,14-6,15-23
- Operator selection option list. R:base users can generate their own menus using command files. The FileGateway is a menu-oriented system in that all options are selected from menus.
  - Menu files ..... 14-19
  - Menu tree ..... 2-28,4-6
  - Nested menus ..... 14-6
  - REPORTS menus ..... 11-3
- Menu design ..... 4-2,14-18,15-23
- menuname ..... 5-8
- Specifies the name of a menu and may be one to eight characters long.
- MESSAGE ..... 5-29
- R:base SET command keyword. If off, R:base messages are suppressed.
- Messages, status or error ..... 15-46
- MIN ..... 8-9
- R:base COMPUTE command modifier. Used to determine the minimum row value for a specific column.



- MKDIR** ..... 5-37  
R:base command. Used to create a new directory.
- Monitor**  
The unit of a microcomputer that contains the display screen.
- MOVE** ..... 15-12  
R:base command. Used to move text from one variable to another.
- MPW password** ..... 6-10  
Password to allow table modification.
- Multiplan**  
Microsoft's spreadsheet program.  
Column orientation ..... 13-31  
Data formats ..... 13-8  
Spreadsheets ..... 13-9  
Row orientation ..... 13-30  
SYLK file conversion with FileGateway ..... 13-30  
UNLOAD ... AS MPLAN ..... 12-8  
What you need to know before conversion ..... 13-30
- Multiple entries per line** ..... 5-10
- Multiple table query** ..... 2-22
- N**
- #n** ..... 15-45  
Specifies a route number where n is 1, 2, or 3 as determined by the SET POINTER command.
- Naming application menus** ..... 14-21
- Naming reports** ..... 11-8
- nchar** ..... 15-12  
Number of characters (see MOVE).
- Nesting program control structures** ..... 15-20
- NEWPAGE** ..... 15-3  
R:base command. Sends a form feed command to the printer or clears the computer screen.
- NEXT** ..... 15-45  
R:base command. Advances an R:base row pointer to the next row as defined by the SET POINTER command.

- NOCHECK** ..... 7-20  
 R:base LOAD command modifier. Same function as the SET RULES OFF command while in load mode.
- NOFILL** ..... 7-20  
 R:base LOAD command modifier. Disables the FILL condition so that missing entries in a loaded file must be entered.
- NULL** ..... 5-29  
 R:base SET command keyword. Allows you to specify the null character. Default symbol is -0-.
- Null value** ..... 5-29,12-14,15-10,15-15  
 Absence of value. The R:base default symbol for a null value is -0-.
- O**
- On-line HELP—[F10]—** ..... 3-4,5-18
- On-line help with report generation** ..... 11-10
- One-to-many database relationships** ..... 2-15
- OPEN** ..... 5-22  
 R:base command. Locates a defined database and makes it available to review and update.
- Opening a database** ..... 3-13,5-22
- Operand** ..... 11-15  
 A column, variable, or value acted upon by an operator and combined with another operand to form an expression. In the expression  $x + y$ ,  $x$  and  $y$  are operands and  $+$  is the operator.
- Operating system**  
 A collection of computer programs that control the overall operation of a computer and perform such tasks as assigning memory to programs and data, controlling the overall input and output of the system.  
     Commands ..... 5-37  
     Requirements ..... 1-2
- Operator**  
 A symbol (+, −, ×, /, %) representing an operation to be performed on one or more columns or variables. Used in R:base in the definition of variables.  
     In arithmetic expressions ..... 11-15  
     In rules ..... 6-12  
     In string expressions ..... 11-21  
     In WHERE clauses ..... 5-15

- ORDER application main menu options ..... 17-3
- ORDER sample application ..... 17-2
- Orientation of spreadsheets ..... 13-9
- OUTPUT ..... 3-36,11-46
  - R:base command. Directs messages and output to the specified output device and, optionally, to a second specified output device.
  - ... filename ..... 3-37,11-47
  - ... KEYBOARD ..... 3-37,11-46
  - ... SCREEN ..... 3-37,11-46
  - ... PRINTER ..... 3-37,11-46
- Output device
  - A device attached to the computer to which commands and data may be sent such as the terminal screen, printer, or disk drive. (See Input device.)
- Output file ..... 3-37,11-47
  - A file to which data and/or commands are written from R:base or the FileGateway. (See Input file.)
- OWNER ..... 6-9
  - R:base command. Used to establish the owner passwords for a database. (See Passwords.)
  
- P**
  
- PACK ..... 5-25
  - R:base command. Used to compact a database, releasing unused space to general use.
- #PAGE ..... 11-20
- Page length for R:base reports ..... 11-44
- Paragraphing in reports ..... 11-26
- paramlist ..... 5-8
  - Parameter list. A list of up to nine values to be passed to a command file.
- PASSWORDS ..... 6-9
  - R:base command. In general, a character sequence used to limit access to a database. R:base supports two password types: Database and Table. Operator access is limited by the password entered with the USER command.
  - MPW ..... 6-10
  - OWNER ..... 6-9
  - RPW ..... 6-10

- 
- Path** ..... 1-5,1-8  
A series of DOS subdirectory names separated by backslashes (\). The path specification tells R:base where to look for files.
- PAUSE** ..... 15-22  
R:base command. Suspends command file execution until the operator responds in the way described in a preceding WRITE command.
- PC-DOS** ..... 1-2  
Operating system for the IBM PC, PC-XT, and PC-AT.
- PFS:FILE**  
A Software Publishing Corporation application product.  
File conversion ..... 13-3,13-51  
What you need to know before conversion ..... 13-51
- PLUS** ..... 5-26  
Same as +. An R:base SET command keyword. Allows plus character to be defined as any other character.
- PLUS command continuation** ..... 5-10  
Plus sign (+) is used to continue a command line. The plus special character is used only as the continuation character. It is not the same as the math operator (+).
- Pointer** ..... 15-45  
Identifier used in the SET POINTER command to move program execution to a specified row.
- PRINT** ..... 3-36,11-45  
R:base command. Prints the contents of a table using a report format.
- Printer control setting** ..... 1-10,11-22
- Printing mailing labels** ..... 11-49,17-10
- Printing reports** ..... 11-46
- procfile** ..... 5-8  
A binary procedure file created by RCOMPILE.
- Program** ..... 15-3  
A series of instructions organized into a procedure that the computer executes.
- Programming control structures** ..... 15-13
- PROJECT** ..... 3-19,10-3  
R:base command. Creates a new table from an existing table and allows you to select the columns to be contained in the new table.

- PROMPT** ..... 5-20,15-27  
R:base command. Used for assistance in developing and executing an R:base command.
- Prompt**..... 5-3  
A symbol or word that appears on the screen to request you to make an entry.
- Prompt blocks** .....15-28
- Q**
- Qualifying data entries** ..... 6-11
- Querying**  
Data ..... 2-21,3-14  
Database Structure ..... 3-17,6-20,13-14
- QUIT** .....15-20  
R:base command. Terminates all WHILE and IF blocks and closes active command files.
- QUOTES** ..... 5-26  
R:base SET command keyword. Allows quotes character to be defined as any other character.
- R**
- R:base 5000**  
Arguments..... 5-7  
Commands..... 5-5  
Description..... iii, 2-2  
Disk contents ..... 1-19  
Editor (see RBEDIT).....15-4  
Keywords..... 5-6  
Main menu ..... 1-11,3-5  
Reserved words..... 5-6  
Specifications..... 1-18  
System requirements..... 1-2,15-5
- RBASE.DAT file**.....1-14,4-9,14-28,15-51
- RBEDIT** ..... 1-12,15-5  
R:base editor that may be used inside R:base with small files or outside R:base with larger files.
- RBEDIT keyboard functions** .....15-5

---

RCOMPILE .....	16-2
The R:base compiler used for compiling R:base command files.	
RD .....	5-37
Abbreviation for RMDIR (Remove Directory).	
REAL .....	6-6,14-12,15-10
Data type representing numbers that may have a fractional value. Range is approximately +9 times 10 to the +37th.	
Decimal point alignment in REPORTS .....	11-55
Recalling a previous input line .....	5-11
Record (See Row.)	
Redefining	
Databases .....	6-15
Report variables .....	11-22
Reentering previous commands .....	5-11
Relation .....	13-7
Another term for table	
Relational database .....	2-2
A database in which data is stored in a set of tables with each table having columns common to each row and having unique data in each row.	
Relational operations .....	10-2
RELOAD .....	5-36
R:base command. Rebuilds a database under a new name and recovers space from deleted rows.	
Reloading the internal database .....	5-36
REMOVE COLUMN .....	6-18
R:base command. Removes a selected column from a selected table.	
REMOVE TABLE .....	6-18
R:base command. Removes a selected table from a database.	
RENAME COLUMN .....	6-19
R:base command. Changes the name of a column.	
RENAME filespec .....	5-37
R:base command. Used to change the name of an operating system file.	
RENAME OWNER .....	6-19
R:base command. Changes the OWNER password.	
RENAME password	
R:base command. Renames modify or read password .....	6-19

- RENAME TABLE ..... 6-19  
R:base command. Changes the name of a table.
- Report  
Columns ..... 3-30,3-32,11-18,11-25,11-29  
Lines ..... 3-33,11-30  
Detail ..... 3-35,11-38  
Footing ..... 3-34,11-32,11-35  
Heading ..... 3-33,11-32,11-35  
Locations ..... 3-32,11-24  
Columns ..... 3-32,11-25  
Variable ..... 3-32,11-25  
Masks for dollars ..... 11-29  
Variables ..... 3-31,11-14
- REPORTS ..... 3-28,11-7  
R:base command. Enter report definition mode. Also the name of the table  
which holds user-generated reports.  
Advanced concepts ..... 11-48  
Changing ..... 11-22,11-29  
Creating ..... 3-28,11-7  
Decimal point alignment for REAL numbers ..... 11-55  
Deleting ..... 11-48  
Design ..... 3-28,11-7  
Editing keys ..... 3-30,11-13  
Features ..... 3-28,3-35,11-4  
Page length ..... 11-44  
Paragraphing ..... 11-26  
Printing ..... 3-36,11-45  
Reports menu ..... 3-29,11-8  
Variables ..... 3-31,11-14  
Width ..... 11-4
- Reserved words, R:base ..... 5-6
- RETURN ..... 15-20  
R:base command. Used in command files to return to the command file that  
called the current command file.
- Reusing the previous input line ..... 5-11
- REVERSE ..... 5-29  
R:base SET command keyword. If off, no reverse video appears.
- RIM ..... 12-6  
Relational Information Management system that operates on a mainframe  
computer.

- 
- RMDIR** ..... 5-37  
 R:base command. Used to remove the designated directory from a designated path.
- Row**  
 The data defined by a set of columns in a table. Some relational databases refer to a row as a record.  
     Description ..... 2-5
- rptname** ..... 5-8  
 Specifies the name of a report and may be one to eight characters long.
- RPW password** ..... 6-10  
 Password used in conjunction with a MPW, allowing read only access to a table.
- RULES** ..... 6-11  
 R:base DEFINE mode command. Used to enter data entry rules. Also the name of the R:base table that holds user-defined rules.
- Rules**  
     Defining ..... 6-11  
     Listing ..... 6-24  
     Operators ..... 6-12  
     Set checking ON or OFF ..... 5-30
- RUN** ..... 15-7,15-48  
 R:base command. Executes a series of commands stored in a command file.
- Running a command file** ..... 15-7,15-48
- S**
- SCRATCH** ..... 5-30  
 R:base SET command keyword. ON locates temporary scratch files on the same drive/directory as the database. OFF locates scratch files on the default drive/directory.
- Scratch Files** ..... 5-30
- \$SCREEN** ..... 16-2  
 Definition line used in a procedure file to indicate the following code is a screen block.
- SCREEN** ..... 5-24  
 Used with the OUTPUT command to direct data to the terminal screen.
- Screen prompts** ..... 5-3
- scrnname** ..... 5-8  
 A defined table or variable form.



scrnrow/scrncol .....	5-8
The terminal screen columns and rows.	
SELECT .....	2-21,3-14,8-2
R:base command. Displays or prints row data from a requested table.	
SEMI .....	5-26
R:base SET command keyword. Allows a semicolon character to be defined as any other character.	
SET .....	5-25
R:base command. Turns conditions off or on. Specifies special characters, date format, user passwords, rules, color, etc.	
SET ERROR .....	15-46
R:base command. Defines a variable to hold error values.	
SET keyword .....	5-27
R:base command used in conjunction with various keywords to control the environment and status of R:base processing.	
SET POINTER .....	15-45
R:base command. Allows you to indicate a specific row on which to perform a function. With NEXT, the row pointer is moved to the next available row meeting the specified conditions.	
SET VARIABLE .....	15-8
R:base command. Sets the data type and/or value of any global variable.	
Setting .....	
Operating conditions .....	5-27
Page length in reports .....	11-44
Printer controls in reports .....	11-22
Special characters .....	5-26
Text wrap locations in reports .....	11-26
Single table query .....	2-21,8-2
SHOW .....	5-32
R:base command. Displays special characters and options.	
SHOW ERROR .....	15-46
R:base command. Shows error code value.	
SHOW VARIABLES .....	5-33,15-11,15-48
R:base SHOW command. Displays the current global variables and their values.	
SORTED BY .....	2-22,3-15,5-13
R:base WHERE clause modifier. Allows sorting of up to ten columns in ascending or descending order.	

Sorting data .....	5-13
Special characters .....	5-26
Specifying operating conditions .....	5-27
Specifying passwords .....	6-9
Spreadsheet .....	13-9
Rows and columns .....	13-9
Specifying boundaries .....	13-11
Totalling fields .....	13-11
Starting FileGateway .....	1-6,1-9
Starting R:base on a floppy disk system .....	1-9
Starting R:base on a hard disk with 256K RAM .....	1-6
Starting R:base on a hard disk with 320K or more RAM .....	1-5
Starting RBEDIT .....	1-6,1-10,15-4
Starting RCOMPILE .....	1-6,1-10,16-5
Starting the Application EXPRESS .....	1-6,1-9,3-6,14-3
Storing application files .....	14-27
String	
A single text field or column. Text strings may be combined to form new text strings.	
Concatenation .....	11-21,15-12
Subtotals in reports .....	11-40
SUBTRACT .....	10-11
R:base command. Creates a new table containing only the rows not in common from two specified existing tables.	
SUM .....	8-9
R:base COMPUTE command modifier. Adds the columns from each row satisfying the command's WHERE clause.	
SUM OF .....	3-31,11-15
R:base variable expression operator used when defining variables for reports. Adds all values for the specified column.	
SYLK Files, Multiplan	
Definition .....	13-32
File conversion .....	13-33
Unloading R:base to SYLK .....	12-8
Symphony .....	13-41
A Lotus Development Corporation spreadsheet application system.	

Syntax .....	5-4
The structure of a command that incorporates the rules by which R:base recognizes the entry as a valid command.	
System configuration .....	1-3,1-7
System requirements for R:base 5000 .....	1-2
System variables #DATE, #PAGE, #TIME .....	11-20,15-13

## T

Table .....	2-2,2-5,10-2,13-7
Representation of storage in a database. Composed of columns and rows. An R:base database can have up to 40 tables.	
Table characteristics of data models .....	2-5
Table forms .....	3-21,7-3
Table lookups in command files .....	15-41
Table passwords .....	6-10
TABLES .....	6-8
R:base command used only in define mode. Used to define a table.	
Tables, linking .....	2-15,2-17
TALLY .....	8-10
R:base command. Outputs the distribution for a column, giving the number of times each value appears in a table.	
Tallying column values .....	8-10
tblname .....	5-8
The name of a table used in command syntax diagrams.	
Terminal	
When used with microcomputers, usually synonymous with computer. When used in the R:base OUTPUT command (OUTPUT TERMINAL), synonymous with OUTPUT SCREEN.	
Testing EXPRESS applications .....	4-9

- TEXT ..... 6-6,14-12,15-10  
 Data type representing alphanumeric data. Strings of text data may be up to 1500 characters long.
- Text editing keys ..... 9-11,15-5
- Text wrap in reports ..... 11-26
- Three-up labels ..... 17-3,17-10
- #TIME ..... 11-20,15-13
- TIME ..... 6-6,11-20,14-12,15-10  
 Data type representing time in hours, minutes, and seconds. The format is hh:mm:ss.
- Totaling column values ..... 8-7
- Totaling fields, spreadsheet ..... 13-11
- Transferring R:base tables to other systems ..... 12-3
- Trapping error messages in command files ..... 15-46
- TYPE ..... 15-24  
 R:base command. Lists the contents of external ASCII files.

## U

- UNION ..... 10-7  
 R:base command. Combines all columns and rows of two existing tables into a new table when the tables have at least one common column.
- UNLOAD ..... 5-34  
 R:base command. Outputs the specified information to the specified output device.
- ... SCHEMA—outputs the database structure ..... 5-35,12-3
  - ... DATA—outputs the data ..... 5-35,12-3
    - ... DATA AS ASCII ..... 12-5
    - ... DATA AS DIF ..... 12-7
    - ... DATA AS MPLAN ..... 12-8
  - ... ALL—outputs both schema and data ..... 5-34,12-3
- Unloading R:base to
- ASCII delimited files ..... 12-5
  - DIF files ..... 12-7
  - Multiplan SYLK files ..... 12-8

- USER** ..... 5-23  
R:base command. See Password. Also an R:base SET command keyword.  
Used to enable or disable the user password.
- USING**  
R:base clause to limit the columns to be used by the command.
- V**
- Value**  
Specific number, character, word, or set of words defined for a column.
- Varformname**  
A one to eight character variable form name ..... 5-8
- Variables**  
Symbols that may assume a succession of values. An R:base variable name may have one to eight characters.  
Changing location in reports ..... 11-29  
Creating ..... 15-9  
Data types for ..... 15-10  
Formats in reports ..... 11-22  
Global ..... 5-31,15-8  
Report ..... 11-22  
System (#DATE, #PAGE, #TIME) ..... 11-20,15-13  
Values ..... 15-11
- Variable forms** ..... 7-3,15-32
- varlist** ..... 5-8  
Specifies a list of variables.
- varname** ..... 5-8  
The name of a global variable used in command syntax diagrams.
- Vector**  
A one-dimensional array. Each element in the vector is a column.
- Viewing data** ..... 8-2
- Viewing database structure** ..... 3-17,6-22
- VisiCalc**  
A Visicorp product. The files may be transferred, in DIF format, to be used in an R:base database.  
Data formats ..... 13-28  
DIF files ..... 13-29  
File conversion procedures ..... 13-27  
UNLOAD ..... AS DIF ..... 12-7  
What you need to know before conversion ..... 13-27

**W**

- WHERE**..... 2-22,3-15,5-14  
 R:base condition clause. Compares two columns in a row or a column to a value.  
     WHERE clause operators ..... 5-15  
     Special use with JOIN command .....10-14
- WHILE. . THEN. . ENDWHILE** .....15-17  
 R:base programming command. Provides conditional processing of commands between the WHILE and ENDWHILE commands until the specified conditions are no longer true.
- WIDTH** ..... 5-30  
 R:base SET command keyword. Sets the standard output width of data sent to the printer.
- Width specification in SELECT command ..... 8-5
- Word processing packages interface to R:base .....12-22  
 (See ASCII delimited files.)
- WordStar** .....12-22  
 A Micropro product. Word processing program for microcomputers. (See MailMerge.)
- Working copy** ..... 1-7  
 A copy of the programs and files required to perform a specific operation.  
 Make a working copy by copying the required programs and files to a formatted disk.
- WRITE** .....15-24  
 R:base command. Used in a command file to display a message at the operator's terminal.

