



“The language is as close to English as any I’ve seen. Programmers skilled in any 3GL will have no trouble. This appeal to both technical and non-technical users is an important characteristic of PROGRESS. It’s a powerful tool in the hands of either kind of user.”

Evan Birkhead, DEC Professional

“The PROGRESS language is so powerful that it redefines the work day. ”

Mike Friedlander, PC Magazine

“With PROGRESS, we were able to complete the same amount of work over a weekend that had taken us two months to accomplish with Informix.”

*Greg Miller, VP Marketing & Sales
Symix Computer Systems Inc.*

“PROGRESS [is an] excellent choice for producing robust applications that may range from simple to complex. The integration between PROGRESS’ data dictionary and procedural language makes it an excellent prototyping tool as well.”

Paul Farr, UNIX Review

“PROGRESS is a programmer’s dream.”

*Pam Young, Applications Manager
New Jersey Transit*

“There is nothing comparable to PROGRESS on the market today at the PC, supermicro, and minicomputer level.”

*Pat DePronio, MIS Director
Marriott Corporation, FSM Division*

Copyright © 1990 Progress Software Corporation

617-275-4500

PROGRESS is copyrighted and all rights are reserved by Progress Software Corporation. This manual is copyrighted and all rights are reserved. This document may not, in whole or in part, be copied, photocopied, translated, or reduced to any electronic medium or machine readable form without prior consent, in writing, from Progress Software Corporation.

PROGRESS® and **PROGRESS FAST TRACK®**
are registered trademarks of
Progress Software Corporation.

Printed in U.S.A.

June, 1990

Product names are a trademark of their respective manufacturers.

Contents

PREFACE	i
THE ORGANIZATION OF THIS BOOK	i
TYPOGRAPHICAL CONVENTIONS	i
OTHER USEFUL PUBLICATIONS	i
CHAPTER 1: MOVING FORWARD WITH PROGRESS	1-1
PATHS THROUGH THE TEST DRIVE	1-1
WHY USE PROGRESS TO DEVELOP APPLICATIONS?	1-3
APPLICATIONS ARE 100% PROGRESS	1-3
QUICKLY PROTOTYPE AND MODIFY APPLICATIONS	1-4
DELIVER HIGH-PRODUCTIVITY TOOLS FOR END USERS	1-4
PROGRESS APPLICATIONS PROVIDE AUTOMATIC RECOVERY	1-4
MULTI-USER CAPABILITY IS BUILT INTO PROGRESS	1-5
PROGRESS PERFORMS	1-6
PROGRESS IS PORTABLE	1-6
PROGRESS SUPPORTS X WINDOWS	1-7
PROGRESS ACCOMMODATES YOUR USERS	1-7
PROGRESS COMPONENTS	1-8
A NOTE ABOUT THE TEST DRIVE	1-13
WHAT'S NEXT?	1-13

CHAPTER 2: WRITING YOUR OWN PROGRESS APPLICATION	2-1
STARTING TO WRITE A PROGRESS APPLICATION	2-2
STEPS FOR WRITING A PROGRESS APPLICATION	2-3
CREATING AN EMPTY DATABASE AND STARTING PROGRESS	2-4
A FEW KEYS TO REMEMBER	2-7
USING THE PROGRESS DICTIONARY TO DEFINE YOUR DATA	2-7
DEFINING FIELDS FOR THE SALES REP FILE	2-12
DEFINING AN INDEX FOR THE SALES REP FILE	2-19
WRITING PROCEDURES TO DO THE APPLICATION'S WORK	2-25
EDITING KEYS	2-25
USING PROGRESS TO ADD AND CHANGE DATA	2-26
USING PROGRESS TO REMOVE DATA	2-37
PRODUCING REPORTS WITH PROGRESS	2-42
CREATING MENUS WITH PROGRESS	2-44
USING PROGRESS SQL	2-46
YOU'VE DONE IT!	2-49
CHAPTER 3: THE MAGIC OF A PROGRESS APPLICATION	3-1
ABOUT THE TEST DRIVE APPLICATION	3-2
STARTING THE TEST DRIVE APPLICATION	3-2
NOW THAT THE APPLICATION IS RUNNING... ..	3-6
THE APPLICATION STRUCTURE	3-7
EXPLORING THE TEST DRIVE APPLICATION	3-10
FILE MAINTENANCE	3-10
DISPLAY A RECORD	3-11
UPDATE A RECORD	3-12
ADD A RECORD	3-15
DELETE A RECORD	3-16
PROCEDURES AS SUBPROCEDURES	3-17

PASSING INFORMATION BETWEEN PROCEDURES	3-18
ORDER ENTRY	3-20
WHAT IF YOUR MACHINE GOES DOWN NOW?	3-30
SIMULATING A SYSTEM FAILURE ON DOS	3-31
SIMULATING A SYSTEM FAILURE ON UNIX	3-32
SIMULATING A SYSTEM FAILURE ON VMS	3-32
SIMULATING A SYSTEM FAILURE ON BTOS/CTOS	3-32
HOW PROGRESS RESTORES THE DATABASE	3-33
RUNNING A REPORT	3-35
PROVIDING INFORMATION TO ANOTHER SYSTEM	3-37
MULTI-USER APPLICATIONS	3-39
HAVE A LOOK AROUND	3-45
CHAPTER 4: ON THE FAST TRACK TO MENUS, FORMS, AND REPORTS	4-1
BUILDING APPLICATIONS WITH FAST TRACK	4-2
STARTING PROGRESS FAST TRACK	4-3
THE MENU EDITOR	4-7
THE SCREEN PAINTER	4-7
THE REPORT WRITER	4-8
THE QBF GENERATOR	4-8
MAINTENANCE	4-8
AND THERE'S MORE...	4-8
A FEW KEYS YOU'LL USE	4-9
CREATING A FAST TRACK MENU	4-10
CHANGING THE CHARACTERISTICS OF A MENU	4-14
ASSIGNING ACTIONS TO MENU OPTIONS	4-17
CALLING A REPORT FROM A MENU OPTION	4-20
CREATING A REPORT	4-22
MODIFYING A REPORT	4-28
SORTING DATA IN A REPORT	4-32
GENERATING A MENU PROCEDURE	4-35
LISTING AN OUTLINE OF YOUR MENU	4-36
YOU'RE NOW ON THE FAST TRACK!	4-36

CHAPTER 5: CAN THERE BE MORE FEATURES? 5-1

THE PROGRESS PROCEDURE LIBRARY 5-2

PROGRESS SQL 5-4

MULTI-DATABASE APPLICATIONS 5-5

DATABASE GATEWAYS – ORACLE AND RMS 5-8

PROGRESS HOST LANGUAGE INTERFACE 5-10

THE HOST LANGUAGE CALL INTERFACE 5-11

CHAPTER 6: TECHNICALLY SPEAKING 6-1

HAVING WHAT IT TAKES 6-1

 UNIX 6-1

 DOS 6-1

 OS/2 6-1

 VMS 6-1

 BTOS/CTOS 6-2

 NETWORK SUPPORT 6-2

PROGRESS COMPONENT SPECIFICS 6-2

WHAT ELSE IS THERE? 6-11

**CHAPTER 7: THE PROGRESS FAMILY OF PRODUCTS
AND SERVICES 7-1**

THE PROGRESS FAMILY OF PRODUCTS 7-1

WHEN YOU NEED SUPPORT 7-8

 ANNUAL MAINTENANCE 7-8

 PROGRESS On-Line! 7-9

A NOTE ABOUT US 7-10

WHAT TO DO NEXT 7-11

APPENDIX A: SAVING DATABASE CHANGES **A-1**

IF YOU DO NOT WANT TO SAVE YOUR CHANGES... A-1

IF YOU DO WANT TO SAVE YOUR CHANGES... A-1

 TO DUMP FAST TRACK DATA A-2

 TO DUMP DICTIONARY DATA A-2

 TO MAKE A COPY OF THE EMPTY DATABASE AND START PROGRESS: A-3

 TO RELOAD DICTIONARY DATA: A-5

 TO RELOAD FAST TRACK DATA A-6

GLOSSARY **Glossary-1**

Preface

THE ORGANIZATION OF THIS BOOK

This book is organized as follows:

TYPOGRAPHICAL CONVENTIONS

This document uses the following typographical conventions:

- **Bold typeface** indicates commands and characters you type. It also emphasizes important points.
- *Italic typeface* indicates a parameter or argument you supply. It also introduces new terms and manual titles.
- `Typewriter typeface` indicates system output and **PROGRESS** procedures. It also highlights file names, field names, command names, and menu options in text.

OTHER USEFUL PUBLICATIONS

The following is a list of publications written by Progress Software Corporation which you may find useful:

PROGRESS Installation Notes

Contains step-by-step instructions for installing **PROGRESS**. Describes the prerequisites and procedures to get **PROGRESS** up and running on your machine.

PROGRESS Language Tutorial

Provides a “how-to” guide to PROGRESS fundamentals, designed for both novice and experienced programmers.

PROGRESS Programming Handbook

Details advanced PROGRESS programming techniques. Provides more detailed information about application development with PROGRESS.

PROGRESS Language Reference

A detailed library of information on a number of PROGRESS topics. Provides descriptions and examples for each statement, function, phrase, and operator in the PROGRESS language.

System Administration I: Environments

Explains the DOS, UNIX, VMS, and BTOS/CTOS concepts required to run PROGRESS, and provides information about running PROGRESS on networks.

System Administration II: General

Describes PROGRESS limits, disk and memory requirements, startup and shutdown procedures, backing up and restoring databases, and PROGRESS utilities. It also provides information about security administration, using multi-volume databases, and Roll Forward Recovery.

Pocket PROGRESS

Lets you quickly look up information about the PROGRESS language or programming environment.

PROGRESS FAST TRACK Tutorial

Provides a “how-to” guide for using PROGRESS FAST TRACK. This manual is designed to lead both a novice end-user and an application developer through the major FAST TRACK features.

PROGRESS FAST TRACK User's Guide

Provides extensive descriptions and examples for the commands and screens in the FAST TRACK application.

The Developer's Toolkit Guide

Explains how to use the PROGRESS Developer's Toolkit, a set of tools used to prepare PROGRESS applications for distribution.

Database Gateways

Provides information about how to use the PROGRESS 4th generation programming language on different relational database management systems other than PROGRESS RDBMS.

3GL Interface Manual

Supplies information about the PROGRESS Host Language Call (HLC), embedded SQL, and using 3rd party software to access the PROGRESS RDBMS.

CHAPTER 1

MOVING FORWARD WITH PROGRESS



CHAPTER 1

MOVING FORWARD WITH PROGRESS

Open up your options! When you choose PROGRESS to build and run applications, you open up the widest range of options provided by any 4GL or database product.

The PROGRESS Test Drive is an evaluation package designed to make the best use of your time, and let you experience the full power and flexibility of the PROGRESS Application Development System.

With over 40,000 licenses worldwide, PROGRESS is one of the industry's leading fourth-generation languages and relational database management systems (4GL/RDBMS). With PROGRESS:

- You can rapidly develop, deploy, and modify critical database applications entirely in PROGRESS, while still maintaining all the control and flexibility found in languages like C or COBOL. The PROGRESS 4GL fully supports ANSI-standard SQL, allowing you to mix SQL and PROGRESS statements in the same procedure.
- You can build and run transaction processing applications across distributed databases. The PROGRESS RDBMS provides such features as automatic crash recovery, multi-user record locking, centralized data validation and user security, and full on-line backups.
- You can port and maintain multi-user transaction processing applications on over 200 hardware platforms, supporting UNIX, XENIX, RISC/ULTRIX, ULTRIX, AIX, A/UX, MS-DOS, OS/2, VAX/VMS, and BTOS/CTOS, as well as a number of network protocols. PROGRESS applications can also simultaneously access data residing in ORACLE databases and Digital Equipment's RMS file structures.

PATHS THROUGH THE TEST DRIVE

For a complete understanding of PROGRESS products and services, we think everyone should read the book from cover to cover, starting on page 1. But we know that may not be your style. So, look at the accompanying chart and decide your optimal learning path for the PROGRESS Test Drive.

What Do You Want?

Read Chapters In This Order:

I want to know about everything that PROGRESS has to offer.	1 2 3 4 5 6 7
I want to write an application of my own right now!	2 3 4 5 6 1 7
I would like to write my own application, but want to see a PROGRESS application in action first.	3 2 4 5 6 1 7
I want to first learn about PROGRESS FAST TRACK, then build my own application.	4 2 3 5 6 1 7
I want a technical overview first, and the background information later.	6 2 3 4 5 1 7

- 1** An overview of PROGRESS: its components, and why it's the right 4GL/RDBMS for you.
- 2** An opportunity to build your own PROGRESS application.
- 3** An exploration of a "real life" application written entirely in the PROGRESS 4GL.
- 4** A quick tutorial on PROGRESS FAST TRACK, that lets you paint screens and menus, write custom reports, and perform ad hoc queries on the database.
- 5** An overview of other PROGRESS features, including the PROGRESS Procedure Library, Database Gateways, embedded SQL, and the Host Language Call (HLC) Interface, and support for ANSI-standard SQL.
- 6** Technical overview of PROGRESS components as well as PROGRESS functionality.
- 7** A description of the PROGRESS family of products and of the support services you can get from Progress Software Corporation.

WHY USE PROGRESS TO DEVELOP APPLICATIONS?

Most likely, you have a number of good reasons for considering PROGRESS:

- You are a developer who is tired of the long development and maintenance cycles so common with traditional languages like C or COBOL, and need to find a way to reduce your application backlog.
- You need to automate critical business functions such as accounting, manufacturing, and distribution, and you want a high-performance RDBMS for on-line transaction processing.
- You heard about the ease of use, high productivity, high performance, and transparent portability that the PROGRESS 4GL/RDBMS offers, and decided to test the product for yourself.

These are some of the many good reasons for evaluating PROGRESS. Why use PROGRESS instead of another application development environment?

APPLICATIONS ARE 100% PROGRESS

First of all, with PROGRESS you can build an entire application in the PROGRESS fourth-generation language. And we don't mean that you work in a "high-level environment" but then have to drop down to C or COBOL to complete the job, as is true with most 4GLs on the market. We mean that the structured, English-like PROGRESS 4GL is a high-level development language that can handle all aspects of your application. The PROGRESS 4GL is much easier to learn and use than traditional languages, because so much of the functionality that programmers require is already built in.

Yet, the PROGRESS 4GL is flexible enough to let you override default features and assume low-level programming control, right down to the keystroke level, if your application requires it.

And the PROGRESS compiler reads SQL as if it were 4GL syntax. This allows you to integrate SQL and the PROGRESS 4GL in the same procedure. You can use SQL for simple querying routines, and the powerful 4GL syntax for complex tasks. With PROGRESS, you get the best of both worlds.

PROGRESS developers never run into a wall when they want to try something outside default parameters. You'll find that PROGRESS doesn't limit the way you access and present your data. PROGRESS gives you the flexibility you need as your application requirements get more demanding.

Of course, if you have invested a lot of development time and effort into building libraries of C routines, you don't have to throw them away. You can call C routines from your PROGRESS application using the PROGRESS Host Language Call (HLC) Interface. The HLC Interface gives you the flexibility to integrate specialized computations or external devices with your PROGRESS application. With PROGRESS, you can read and write data to and from bar code readers, video disk players, and other data sources.

PROGRESS applications can also export DIF, SYLK, and ASCII files to spreadsheets, word processing systems, and other office automation software.

QUICKLY PROTOTYPE AND MODIFY APPLICATIONS

Developing database applications with the PROGRESS 4GL is fast. And with PROGRESS FAST TRACK, developers can rapidly prototype and modify the user interface for their applications. FAST TRACK lets you paint screens, menus, and reports through a combination of menu choices, simple commands, and point-and-pick techniques. And FAST TRACK translates the screens, reports, and menus you create back into complete PROGRESS 4GL procedures. You can then integrate these procedures into other PROGRESS applications as well.

FAST TRACK itself is written in the PROGRESS 4GL, demonstrating the vast power of the language.

DELIVER HIGH-PRODUCTIVITY TOOLS FOR END USERS

FAST TRACK is a powerful tool for your end users as well. Together with your PROGRESS application, end users can use FAST TRACK to create and modify custom reports that they can either print out or review on-line. FAST TRACK also lets your users perform ad hoc queries on the database, without any risk of corrupting data.

PROGRESS APPLICATIONS PROVIDE AUTOMATIC RECOVERY

You've probably experienced power failures before. Are you tired of losing all your data or manually reconstructing your database rather than lose hours of data

entry? PROGRESS has automatic Roll Back Recovery that preserves database integrity through most hardware, software, and environmental failures. This recovery includes fully automated backout of all incomplete transactions upon system restart.

Many database products provide some sort of protection against system and power failures. These recovery techniques work as long as your media isn't damaged. Even if you back up your database every day, a loss of media can be a considerable setback. PROGRESS gives you the choice of also protecting your database from losses due to damaged media. If you use PROGRESS' Roll Forward Recovery, you can restore all the completed transactions you would otherwise have had to re-enter, up to the point the media became damaged.

MULTI-USER CAPABILITY IS BUILT INTO PROGRESS

If you are building a multi-user database application, you need to consider a number of variables. What if one user requests a file that is currently being updated by another user? You don't want one user tying up an entire file while another user is waiting for a record in that file. At the same time, you don't want multiple users to be changing the same record simultaneously. How could you be sure which set of changes was actually stored in the database?

None of this is a problem with PROGRESS.

An application developed in PROGRESS has built-in multi-user capability. PROGRESS' automatic concurrency features support simultaneous access and updates by many users without complex record locking protocols. Logical transactions involving multiple record updates are fully supported, and PROGRESS allows transaction backout when user tasks conflict. All this ensures data integrity in your PROGRESS database application. The PROGRESS 4GL also gives you the control to override the automatic file and record locks whenever your application requires it.

If your application is running across distributed databases, you may often have identical records on more than one database. So how can you be sure that a record update will be executed across all databases? The PROGRESS RDBMS uses a two-phase commit process that checks to see that all databases are active and have received the update request. When all the databases send back the correct confirmation, the update is committed to each database. If a database does not confirm that it is active, then the record update is cancelled, and all databases are restored to their prior state.

PROGRESS PERFORMS

In addition to being easy to learn, use, and modify, the PROGRESS 4GL/RDBMS gives you performance gains over other application development languages and database management systems.

The PROGRESS relational DBMS uses compressed B-tree indexing, variable-length data storage, and a multi-threaded architecture to ensure high application performance.

And PROGRESS automatically compiles a procedure before running that procedure for the first time. PROGRESS then holds that compiled procedure in memory for the remainder of your PROGRESS session. On subsequent runs, this compiled version of the procedure is used, making execution more efficient than if the source code had to be compiled or interpreted again. PROGRESS procedures can also be compiled and saved for use in later sessions, maximizing the performance of your complete application.

To maximize application performance, the PROGRESS relational database uses a client-server architecture that separates front-end application processing from the actual database processing. This makes efficient use of your available processors, and maintains a high transaction throughput even as you add more users to your system.

PROGRESS IS PORTABLE

Suppose you develop an application to run on an MS-DOS-based PC. Then, one of your users wants to run that application with multiple users on a UNIX system, or on a VMS system, or perhaps a LAN. What do you do? Rewrite the entire application for the new environment? Tell the user it just isn't possible? Once again, the PROGRESS answer is a simple one.



Because you can write your entire application in the PROGRESS 4GL, PROGRESS gives you true application portability and connectability: you can port PROGRESS code, without change, across over 200 hardware platforms, supporting all of the most widely used operating systems, as well as a number of LAN protocols.

PROGRESS applications can also be distributed across many of these same environments in a heterogeneous network; for example, DOS and XENIX applications can access a central database on a larger UNIX machine. This offloads the front-end processing onto the smaller machines and allows the larger CPU to act as the database server.

PROGRESS SUPPORTS X WINDOWS

With the support of X Windows for PROGRESS V6, Progress Software provides a graphic user interface for PROGRESS applications. PROGRESS developers can now create PROGRESS applications that work in windows and that use many of the user interface tools (mouse, icons, etc.) supplied in this window system. You can use any X Windows manager (Motif, OpenLook, etc.) to enforce general window display characteristics for PROGRESS applications. With X Windows, you can run several PROGRESS applications in different windows on a single screen display using a mouse and keyboard simultaneously.

PROGRESS ACCOMMODATES YOUR USERS

Imagine this situation: You've used the PROGRESS 4GL to develop a complete application. Two kinds of users are going to be running your application:

- *The Traditional End User*

This user is interested only in running your application and will not be using PROGRESS to write procedures, compile custom reports, or perform ad hoc queries on the database.

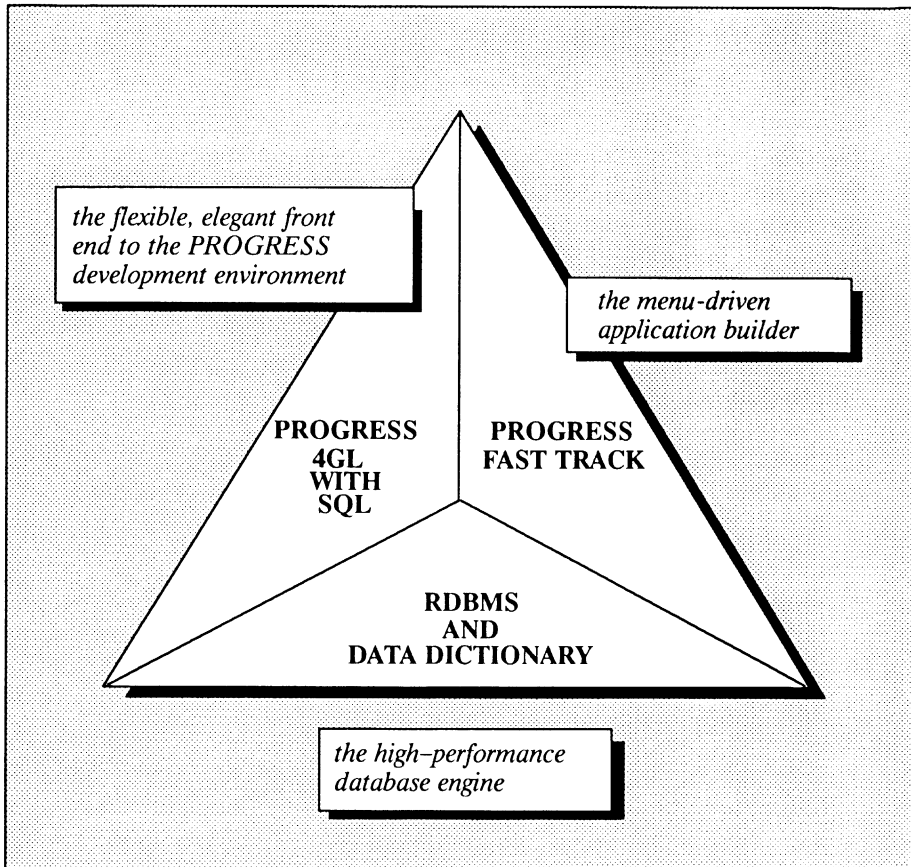
- *The Advanced User*

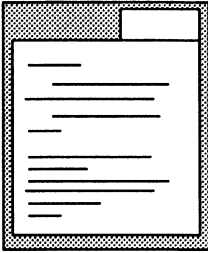
This user wants to run your application but also wants to use PROGRESS to create his or her own own database queries or ad hoc reports. The advanced user won't be taking advantage of all of PROGRESS' application development tools, but will be doing work other than just running your application.

PROGRESS is well-suited to the application developer, the power user, and the end user. You can allow different kinds of users to see as much or as little of PROGRESS, and as much or as little of your application, as is necessary. You can purchase PROGRESS products with various levels of functionality, depending on the needs of your users. Let's take a closer look at the main components of the integrated PROGRESS Application Development System.

PROGRESS COMPONENTS

PROGRESS consists of a tightly integrated set of components that directly relate to each part of your application.





THE PROGRESS FOURTH-GENERATION LANGUAGE

Once you start programming in the PROGRESS 4GL, you'll never go back to your old language. All the control of a traditional structured programming language is built into PROGRESS, so you never need to code in conventional third-generation languages like C, BASIC, COBOL, or PASCAL. And while the PROGRESS 4GL offers you all the power and flexibility of a 3GL, PROGRESS also offers you superior productivity.

As you begin the PROGRESS Test Drive, you will see that the PROGRESS 4GL is an elegant, English-like development language that lets you quickly begin producing useful, transaction-based applications. The PROGRESS 4GL supports ANSI-standard SQL, and includes an integrated library of functions and operators as well as tools for arithmetic and date computation, string manipulation, and system and record status control.

The full-screen, interactive PROGRESS editor checks your syntax, flags any incorrect words or phrases in your procedure, and gives you a message with a possible solution. Of course, you can move, delete, and copy text within a single procedure or from one procedure to another. And a single keystroke lets you test a PROGRESS procedure or retrieve the last procedure you ran. The PROGRESS editor is fully integrated with the PROGRESS environment, but you may still use your own favorite editor if you wish.

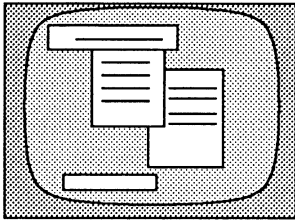
On-line Help is available to you from not only the editor but from anywhere in a PROGRESS session as well. On-line Help gives you detailed information about PROGRESS statements, functions, keywords, operators, error messages, keyboard definitions, PROGRESS limits, and your data definitions.

PROGRESS makes sure your procedures run fast. It automatically compiles the procedure once, yielding compact and efficient code that ensures rapid execution on all subsequent runs of the procedure. Of course, you get all of the normal language features such as the ability to run one procedure from another. But you'll also find many other unexpected features: high-level file and record manipulation

commands, automatic error handling and recovery, automatic lock handling for concurrent file access, the ability to override this automatic error and lock handling with some simple statements, and the ability to call routines written in C.

The PROGRESS 4GL can easily communicate with programs written in third-generation languages such as C or COBOL.

In addition to accessing data in the PROGRESS relational database, the PROGRESS 4GL can also read and write data from Oracle and RMS databases. Your PROGRESS application can run on top of Oracle, RMS, and PROGRESS databases simultaneously.

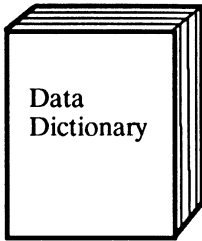


PROGRESS FAST TRACK

While the PROGRESS 4GL has a comprehensive set of defaults that automatically handle screen formatting, you can also take control with the PROGRESS 4GL to customize your screens and reports. And with PROGRESS FAST TRACK, a menu-driven prototyping and reporting tool, you can paint custom screens, menus, and reports without having to write any procedures.

Through a combination of menu choices, simple commands, and point-and-pick techniques, FAST TRACK lets you quickly create screens, menus, overlaid windows, and reports. Once you have designed your desired application screen, FAST TRACK translates the end product into a PROGRESS 4GL procedure that you can use as the basis of new applications or to enhance existing applications.

In addition to creating menus, screens, and reports, FAST TRACK can help make the end users of your PROGRESS application more productive. Delivered along with a PROGRESS application, FAST TRACK lets users with minimal experience create and modify custom reports and perform ad hoc database queries. With FAST TRACK, your users can maximize the power and flexibility of your database application, without the danger of inadvertent data updates.



THE PROGRESS DATA DICTIONARY

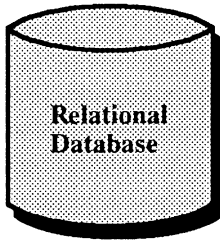
You use the menu-driven PROGRESS Data Dictionary to store all of the information about your database structure, including the names of files, descriptions of fields, definitions of indexes, and specification of index components. The Data Dictionary also holds descriptions of default settings for field formats, columns labels, side labels, initial values, file and field validation, and context-sensitive help messages.

Database definitions are stored dynamically, permitting persons with proper security to make enhancements and modifications to the database structure. With proper authorization, you can add, change, or delete files, fields, and indexes without dumping and reloading the database.

The Data Dictionary provides a full complement of data types to give you flexibility: character, integer, decimal, logical, and date. Arrays of any data type are also supported. You can define multiple indexes per file and multiple fields per index. PROGRESS stores these indexes using a high performance B-tree structure, so your application makes efficient use of disk space and processor resources.

You can define separate read and write protection at both the file and field level. This gives you extensive database security capabilities. PROGRESS lets you define validation at the field level to ensure that only valid data is entered into the database. Once you define validation for a field, that validity check automatically is used for any other procedures that require data entry with that field. And PROGRESS provides full referential integrity; you can define validation on the file level to protect against inadvertently deleting a record on which other records depend.

Of course, if you require it, you can override field and file validations at any time using the PROGRESS 4GL.



THE PROGRESS RELATIONAL DATABASE

The PROGRESS DBMS is fully relational. You can easily and efficiently relate one file (or table) to another and work on many files at one time. You can have up to 1,023 files and 1,023 indexes per database. Each record (or row) can contain an unlimited number of fields (or columns) within a 32,000 byte maximum record size. Because PROGRESS stores fields using a variable-length format, you get optimal storage and retrieval of your data.

Furthermore, PROGRESS databases are not constrained to a single physical volume. They can span multiple disk drives so that your database can continue growing even after it has filled a single volume. Storing PROGRESS on multiple disk drives lets you further enhance performance of your application. PROGRESS applications also support distributed database configurations, allowing you to access and update records from multiple databases simultaneously.

Although the PROGRESS 4GL and RDBMS interact seamlessly, there is an alternative method to reading and writing the PROGRESS database. You can embed SQL statements into C applications, and access the PROGRESS database through the PROGRESS HLI (Host Language Interface). These embedded SQL statements can incorporate the requested data back into the C application. So if you already have a significant investment in C, you can easily modify these programs to run on the PROGRESS RDBMS, without having to rewrite the whole application in the PROGRESS 4GL.

The PROGRESS RDBMS supports full on-line backups, so you don't have to make your users log out in order to bring down the application.

With Roll Back Recovery, the PROGRESS database manager automatically ensures the integrity of transactions involving multiple record updates. That is, PROGRESS makes sure that such transactions are either carried out completely or not at all: nothing is left in a partially finished state.

If something goes wrong during a transaction, the database manager automatically restores the updated records to their state prior to the transaction. In addition, if

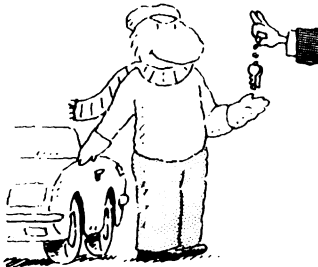
there are multiple users running a PROGRESS application, the database manager automatically handles record locking to be sure that the integrity of the database is not compromised.

For further insurance, you can protect your database from disk failures by using Roll Forward Recovery. On most systems, if a physical disk volume is damaged, you install your most recent backup. However, you will lose the completed transactions that were made since that last backup, which often means long hours of re-entering those transactions. When you use Roll Forward Recovery, you maintain a copy of all transactions made since your last backup on a separate disk. You can then run those transactions again, restoring your database to the point it was at when the media was damaged.

A NOTE ABOUT THE TEST DRIVE

The Test Drive is a full evaluation version of the PROGRESS application development system. This means that you have access to the full power of the PROGRESS 4GL/RDBMS with FAST TRACK, limited only by the number of times you can use the same database. You can run the Test Drive 10 times with the same copy of a database. If you need additional sessions, refer to the instructions in Appendix A before exiting your tenth session.

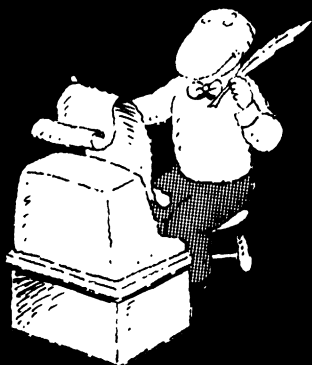
WHAT'S NEXT?



Ready to see what PROGRESS can do for you? Let's go on to Chapter 2 where you'll build an application with PROGRESS!

CHAPTER 2

WRITING YOUR OWN PROGRESS APPLICATION



CHAPTER 2

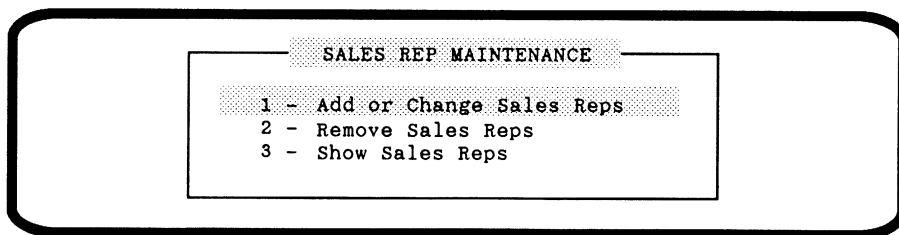
WRITING YOUR OWN PROGRESS APPLICATION

If you're like many of our users, you want to start writing PROGRESS applications right away. In this chapter you begin. You're going to write a small part of the Test Drive application.

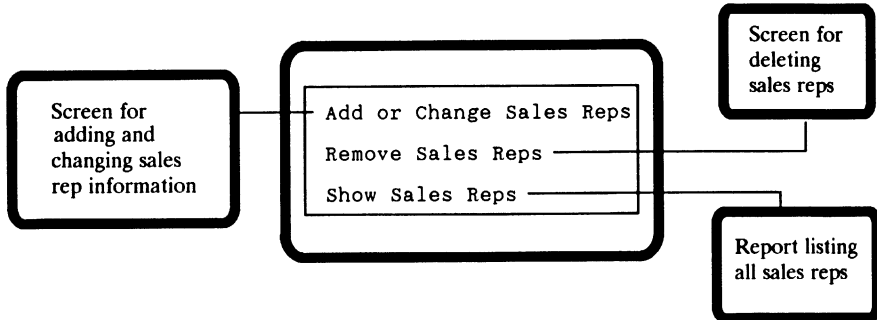
The Test Drive application has been developed in PROGRESS for a fictitious company called All Around Sports. All Around Sports is a sporting goods distributor. Their PROGRESS application helps them keep track of customer orders and inventory levels. Although the company itself is fictitious, the problems that they must deal with to keep up with a growing business, and the solutions that PROGRESS provides, are real.

You're going to write the portion of the application that maintains a database file called the Sales Rep file. This file stores information about the sales people at All Around Sports, including their names and the regions for which they are responsible.

You will write a *procedure* to display the following menu:

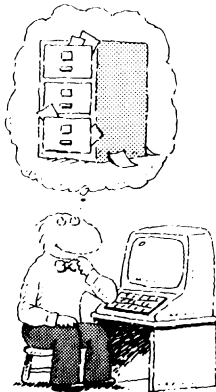


A procedure is simply a portion of a PROGRESS application that performs a specific task. You will also write the procedures that perform each menu choice. Although you can probably guess, the following figure shows what each menu choice does.



Now that you have an idea of what kind of an application you'll be writing, what do you do next?

STARTING TO WRITE A PROGRESS APPLICATION



The best way to learn how to develop applications in PROGRESS is to imagine for a moment that you did not have PROGRESS or even a computer. You decide you want to maintain information about sales reps. What would you do?

First, you would determine the information about each sales rep you want to maintain, such as the names of all the sales reps and the yearly quota for each. Next, you would organize this information, most likely using paper forms.

You would probably store these forms in a file. Deciding what kinds of information you want to store in your database is called *defining the data*.

Once you defined the data, you would set up some kind of system for accessing the sales rep data. In PROGRESS, you use procedures to access data in your database.

STEPS FOR WRITING A PROGRESS APPLICATION

Before you can write a PROGRESS application:

- *Install PROGRESS and the Test Drive*

Before starting PROGRESS, you must install PROGRESS and the Test Drive application according to the instructions in the *Installation Notes* included with your Test Drive package, if you haven't installed them already.

- *Get into your working directory.*

Be sure you are in the directory from which you want to run the Test Drive application. This directory should be a directory other than the one in which the Test Drive software is installed.

To build your application using PROGRESS, you will follow these steps:

- *Create an empty application database and start PROGRESS.*
- *Describe the data.*

To figure out how to describe the data, look at the sales rep information you want to maintain:

- The initials of the sales rep
- The name of the sales rep
- The region the sales rep is responsible for
- The sales rep's job title
- The sales rep's annual budgeted sales quota
- Date of hire

To tell PROGRESS how you want to define your application data, you use the PROGRESS Data Dictionary.

- *Analyze the task.*

Think through all the steps required to perform the work of the application. Since you already have a rough idea of the application you are about to write,

you know that you want to be able to use the application to add, change, delete, and display information about sales reps. We will look more closely at the design of each of these a little later in this chapter.

- *Write and test procedures.*

To write your procedures, you use the PROGRESS procedure editor, where you enter statements in the PROGRESS fourth-generation language (4GL). You can also test your procedures from the editor. With a single keystroke, you can run your procedures to see how they work.

Now, let's begin!

CREATING AN EMPTY DATABASE AND STARTING PROGRESS

First, you must log in to your system. If you are using DOS or OS/2, you probably just need to turn on your computer. If you are using UNIX, VMS, or BTOS/CTOS, type your userid and password, if necessary.

Whether you are running UNIX, DOS, OS/2, VMS, or BTOS/CTOS, be sure you are in the directory from which you want to run the Test Drive. This directory should be a directory other than the one in which the software is installed.

To create your own database, you type a command in response to your operating system prompt. Refer to the following table and type the command for the operating system you are using.

Operating System	To Copy the Application Database
UNIX, DOS, and OS/2	proddb mydrive empty
VMS	PROGRESS/CREATE mydrive empty
BTOS/CTOS	PROGRESS Create Database New Database Name mydrive Copy From Database Name empty

After you type this command, press `RETURN` if you are using UNIX, DOS, OS/2 or VMS. (If you are using a PC, use `ENTER` or `↵` when you see `RETURN` in this book.) If you are using BTOS/CTOS, press the key labeled GO on your keyboard.

PROGRESS displays a message telling you that it has created a new database called mydrive, which is a copy of the system “empty” database. The empty database contains PROGRESS system information but has no application data or data definitions.

To start PROGRESS using the mydrive database, refer to the following table. Type the command for your operating system.

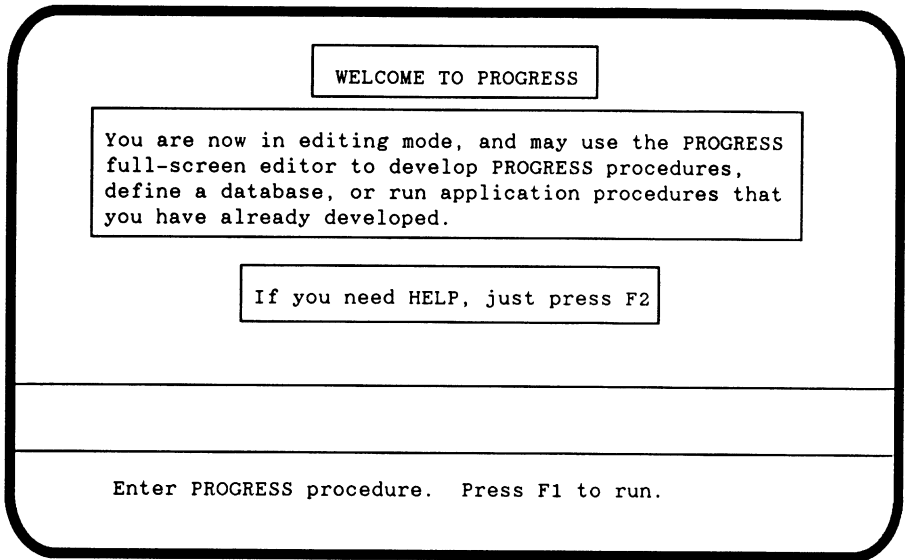
Operating System	To Start PROGRESS
UNIX, DOS, and OS/2	pro mydrive
VMS	PROGRESS mydrive
BTOS/CTOS	PROGRESS 4GL [Options] -1 mydrive

If you are using UNIX, DOS, or VMS, press . If you are using BTOS/CTOS, press the key labeled GO on your keyboard.

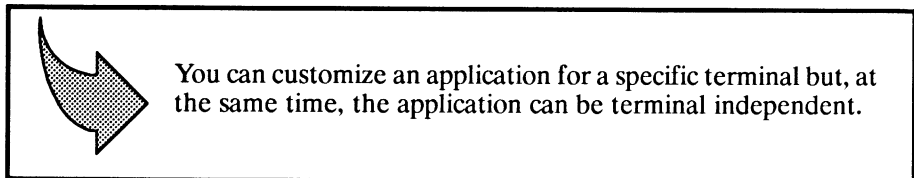
PROGRESS displays a banner followed by this message at the bottom of the screen: “You may use PROGRESS on this database 9 more times.” As you know, the Test Drive is a demonstration version of PROGRESS. You have the full functionality of the PROGRESS software, but you can run the Test Drive only ten times with the same database.

Each time you start PROGRESS, you receive a message that tells you how many sessions you have left with the current database. If you want another ten sessions with the Test Drive, you can make another copy of the empty database and start again from scratch. Or, if you prefer, you can save the data and data definitions you create in this chapter and use them in your next ten-session Test Drive. Appendix A explains the steps you take to save data and data definitions.

Press and you’ll see this welcome screen:



This welcome screen differs slightly from terminal to terminal so your screen may not look *exactly* like this one. PROGRESS runs on many different kinds of machines and on many more different kinds of terminals. But you don't have to worry about creating separate versions of your application for each machine or terminal on which it will run.

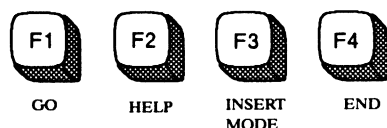


Now you're in PROGRESS. What next? The next step in building a PROGRESS application is to design and describe your application data. You use the PROGRESS Data Dictionary to do this.

Just one more thing before you start up the Dictionary - you need to know a bit more about function keys.

A FEW KEYS TO REMEMBER

You will use several special keys to get around the PROGRESS Test Drive. Here are a few of those keys:



At the back of this book, you'll find a tear-out reference card that shows the special keys you can use in PROGRESS. The reference card shows the keys used for both editing and executing a PROGRESS application.

If your terminal has PF keys instead of function keys, use those keys instead of the ones shown on the reference card. For example, use PF1 instead of F1. If your terminal has neither PF keys nor function keys, use the equivalent control keys shown on the reference card. For example, **CTRL**-X is the same as F1 on most terminals.

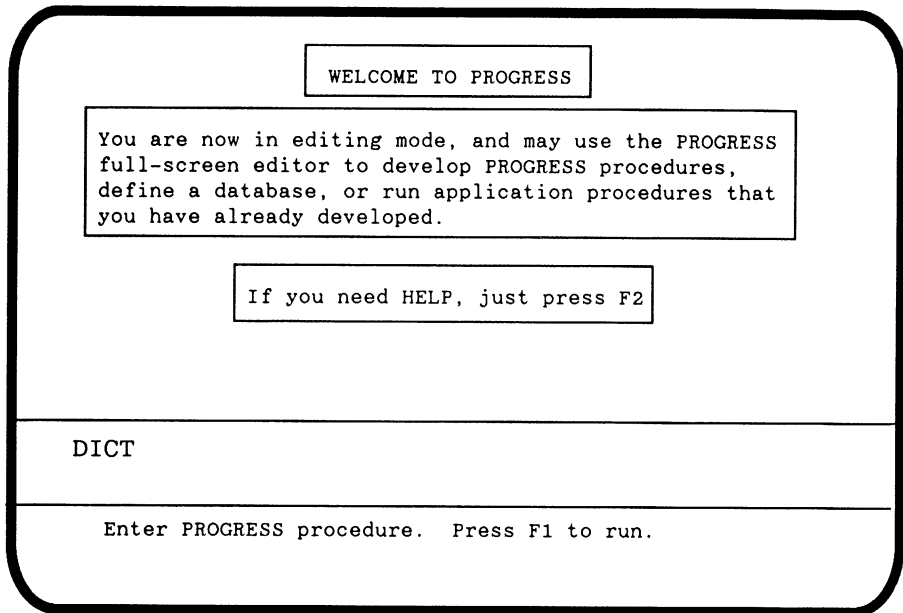
If you are using DOS or OS/2, note that the function keys F11 through F16 correspond to ALT-F1 through ALT-F6.

Whenever you need to press a key, you'll see the name of the key followed by the label of the key as it probably appears on your keyboard (some keyboards may be slightly different). For example, we may tell you to press **GO** (F1). This means press the key labeled F1 on your keyboard. This tells PROGRESS to execute the procedure in the editor or accept the values entered in a menu.

OK, *now* you can start the Dictionary.

USING THE PROGRESS DICTIONARY TO DEFINE YOUR DATA

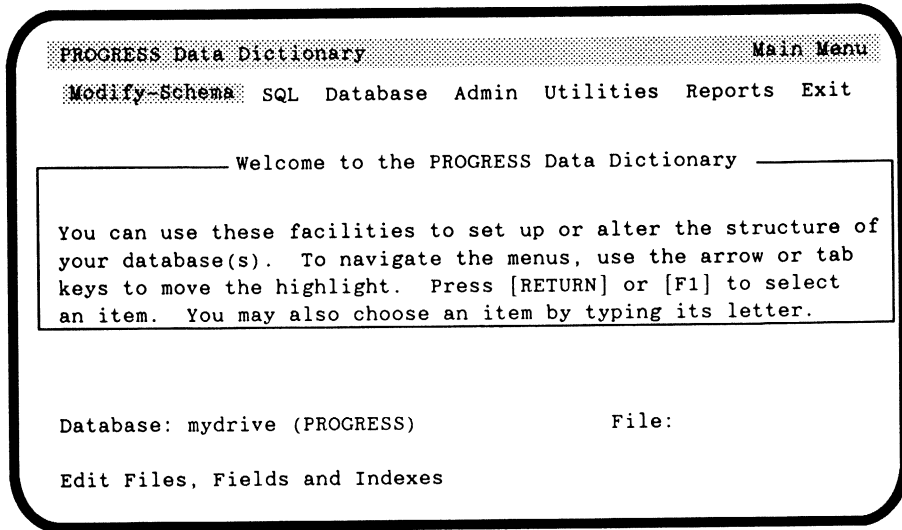
Your cursor is between the two horizontal lines that span the bottom of the screen. To start the Dictionary, just type **DICT** (in either uppercase, lowercase, or a combination of both):



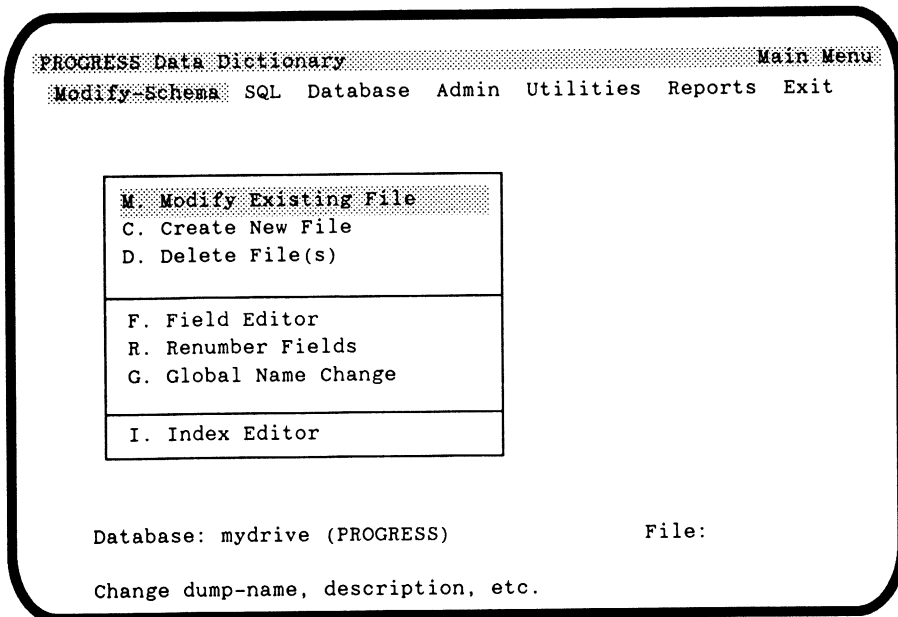
The DICT statement tells PROGRESS to run the Data Dictionary. Press (F1) to enter the DICT statement.

You can also access the Data Dictionary from the editor at any time by pressing (F2) for Help and then selecting "d" from the Help menu.

When you start the Dictionary, you see this next screen. This is the Data Dictionary's Main Menu.



Since you want to create new data definitions for the sales rep information, press **RETURN** or type an **m** to choose Modify - Schema. The following screen appears:



You can now choose from any of the menu items, by moving your cursor to that item and pressing **[RETURN]**. You want to create a new file, the salesrep file, so move your cursor down one line to C. Create New File and press **[RETURN]**. The following screen appears:

```
PROGRESS Data Dictionary                                Create New File
MODIFY-SCHEMA  SQL  Database  Admin  Utilities  Reports  Exit
-----
File Name:                                           Hidden: no
File Type: PROGRESS (NEW FILE)                       Frozen: no
File Owner:                                           File Number: ?
Dump Name: ?           (unique name for data dump into .d file)

Description:

Enter the message to be displayed for disallowed deletions.
      ValMsg:

Enter an expression which must be TRUE to allow record deletions.
      ValExp: ?

-----
Database: mydrive (PROGRESS)                          File:

Enter data or press F4 to end.
```

You can now fill in the information about your file, beginning with file name. The cursor is already positioned at the File Name: prompt, so you can simply type in **salesrep**. Press **[GO]** (F1) to skip over the remaining fields and create the new file. You now see the following screen:

```
PROGRESS Data Dictionary          Create New File
MODIFY-SCHEMA  SQL  Database  Admin  Utilities  Reports  Exit
_____ Currently Defined Fields _____

NextPage  PrevPage  Add  Modify  Delete  Copy  GoIndex  SwitchFile
Browse  Undo  Exit
_____ Total Fields: 0

Database: mydrive (PROGRESS)          File: salesrep

See the next page of fields.
```

Each piece of data about a sales rep, such as initials or sales region, is called a *field*. You'll use this menu to define the fields for the salesrep file.

To choose an option from this menu, you can use either of the following methods:

- Use the arrow keys to highlight the option you want and then press `RETURN`.
- Type the letter of the option.

You want to add the fields that make up the salesrep file, so choose the third option, Add.

The Dictionary displays the following screen where you can enter data about the fields in the file. The data you enter for the fields in a file makes up the *record* for the file.

```

PROGRESS Data Dictionary          Create New File
MODIFY-SCHEMA  SQL  Database  Admin  Utilities  Reports  Exit
-----
                          Currently Defined Fields
-----

Field-Name: _____ Data-Type:
Format:          Extent:
Label:          Decimals:
Column-Label:   Order:
Initial:        Mandatory: (Not Null)
Component of-> View:  Index:  Case-sensitive:
Valexp:
:
:
:
Valmsg:
Help:
Desc:

-----
NextPage  PrevPage  Add  Modify  Delete  Copy  GoIndex  SwitchFile
Browse  Undo  Exit
-----
Total Fields: 0

Database: mydrive (PROGRESS)          File: salesrep

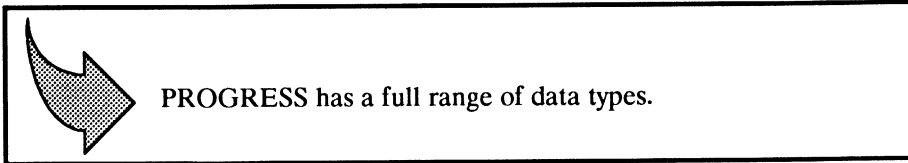
Enter data or press F4 to end.
    
```

DEFINING FIELDS FOR THE SALES REP FILE



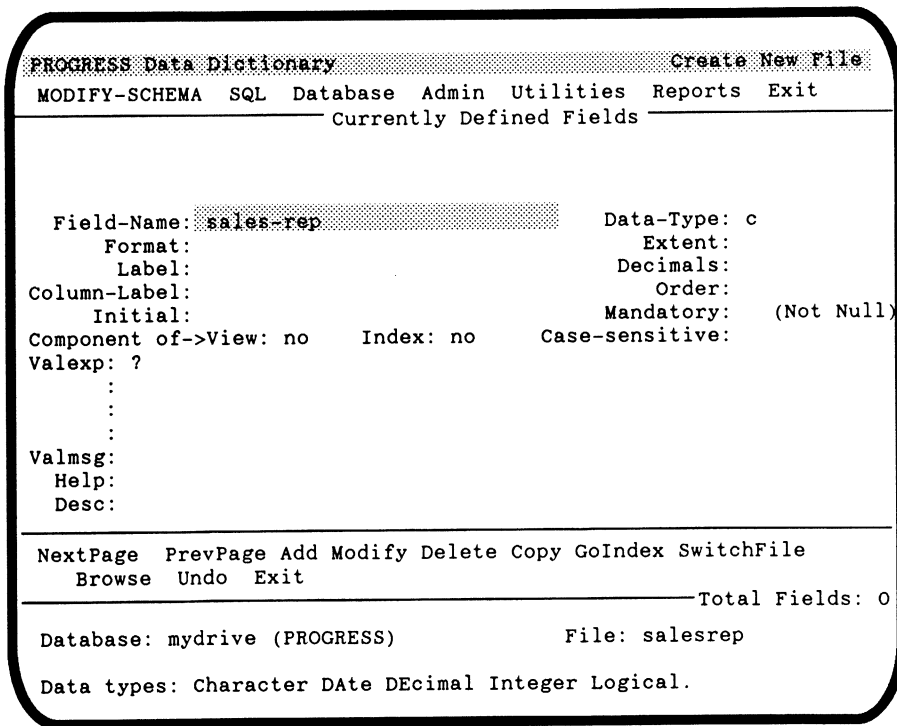
Start by defining the field that will contain the sales rep's initials. Let's call this field sales-rep. Type **sales-rep** as the Field-Name and press **RETURN** to go on to the next field characteristic. The next area that is underlined is the Data-Type.

The *data type* is a label assigned to a field that determines what type of data that field can store. PROGRESS supports five data types: character, integer, decimal, date, and logical.



The information in the sales-rep field consists of the sales rep's initials, which are all alphabetic characters. Therefore, you want to define the sales-rep field to be a character data type.

To supply the data type, you can spell out the word **character**, or you can abbreviate it by just typing **c**.



When you have entered the data type, press **RETURN**. The Dictionary automatically fills in default values for several of the remaining field characteristics.

```

PROGRESS Data Dictionary                               Create New File
MODIFY-SCHEMA  SQL  Database  Admin  Utilities  Reports  Exit
-----
                          Currently Defined Fields

Field-Name: sales-rep                               Data-Type: character
Format: x(8)                                         Extent:
Label: ?                                             Decimals:
Column-Label: ?                                     Order: 10
Initial:                                             Mandatory: no (Not Null)
Component of->View: no   Index: no   Case-sensitive: no
Valexp:
:
:
:
Valmsg:
Help:
Desc:

NextPage  PrevPage  Add  Modify  Delete  Copy  GoIndex  SwitchFile
Browse  Undo  Exit

-----Total Fields: 0
Database: mydrive (PROGRESS)           File: salesrep
Enter data or press F4 to end.

```

You can change these values if you want. Let's assume you want to change the Format characteristic.

The format of a field describes the way that field will be displayed or printed. It does not affect the way PROGRESS stores data. All data is stored variable length, taking up only as much room as it needs. The format only affects the way data appears on the screen. As a default, PROGRESS allows eight spaces for all character fields unless you specify otherwise. But, since you know that the sales rep's initials will take a maximum of three spaces, you can change the format to x(3).

To change the format to x(3), use the right arrow key to move the cursor onto the 8 at the Format : line. Type a 3 and press **RETURN**.

What if you decide that, in certain parts of your application, you want to display the data differently than is indicated in the Dictionary format? Don't worry — you can override the format, within the application, any time you want.

Your cursor is now at the Label : prompt.

The Label of a field determines the label that PROGRESS uses for displaying the field while the application is running. If you do not specify a label, PROGRESS automatically uses the field name as the default label. In this case, PROGRESS would use “sales-rep.” To make the field more readable, let’s use the label “Sales Rep” instead. Type **Sales Rep** as the label and press .

You can also define a column label in the Column-Label field. You use this option when you want a label to appear “stacked” or in columnar form.

What if you decide that in a certain part of your application, you want the label to be different, or maybe that you don’t want any label at all? No problem — you can easily override the label in any procedure.

Press repeatedly to skip over all of the other characteristics until the cursor gets to the Help characteristic. The text you put on this line is displayed by PROGRESS at the bottom of the screen every time data is to be entered into the sales-rep field. Type **Please enter the Sales Representative’s initials** and press .

Here is how the screen looks now:

```

PROGRESS Data Dictionary          Create New File
MODIFY-SCHEMA  SQL  Database  Admin  Utilities  Reports  Exit
-----
                          Currently Defined Fields

Field-Name: sales-rep          Data-Type: character
Format: x(3)                   Extent:
Label: Sales Rep              Decimals:
Column-Label: ?               Order: 10
Initial:                       Mandatory: no (Not Null)
Component of->View: no        Index: no    Case-sensitive: no
Valexp:
:
:
:
Valmsg:
Help: Please enter the sales representative's initials
Desc:

NextPage  PrevPage Add Modify Delete Copy GoIndex SwitchFile
Browse  Undo  Exit

-----Total Fields: 0
Database: mydrive (PROGRESS)          File: salesrep

Enter data or press F4 to end.
    
```

For now, you can leave the rest of the field characteristics just as they are. Press **GO** (F1) to tell the Dictionary that you have finished defining characteristics for the sales-rep field. After you press **GO**, a message appears at the bottom of the screen to indicate that the field was added successfully. A blank field form also appears so you can add more fields. If you are done, press **END** (F4) and the fields you added are saved as a part of your file. After you press F4, PROGRESS lists the sales-rep field as being currently defined.

If, after defining a field, you realize you made a mistake in the definition of that field, just enter the name of the field in the Field-Name area and press **GO** (F1). The field definition appears so that you can change the field's characteristics. You cannot, however, change the data type. If you need to change the data type of a field, press **END** (F4) to return to the Field Editor menu and type **d** to delete the field. Then type **a** and define the field again.

The following are the rest of the definitions for the salesrep file. Please enter them. To get back into the field form, type **a** for Add.

```
Field-Name:  slsname                Data-Type: character
             Format:  x(30)           Extent: 0
             Label:  Name             Decimals: ?
Column-label:                Order: 20
             Initial:                Mandatory: no (not null)
Component of -> View:no  Index:no    Case-sensitive: no
Valexp:
Valmsg:
Help: Please enter name of sales representative (last name first)
Desc:
```

After entering all the information for the slsname field, press **[GO]** (F1).

```
Field-Name:  slsrgn                Data-Type: character
             Format:  x(8)           Extent: 0
             Label:  Region          Decimals: ?
Column-label:                Order: 30
             Initial:                Mandatory: no (not null)
Component of -> View:no  Index:no    Case-sensitive: no
Valexp:
Valmsg:
Help: Please enter the sales region covered by this representative
Desc:
```

After entering all the information for the slsrgn field, press **[GO]** (F1).

```
Field-Name:  slstitle              Data-Type: character
             Format:  x(8)           Extent: 0
             Label:  Title           Decimals: ?
Column-label:                Order: 40
             Initial:                Mandatory: no (not null)
Component of -> View:no  Index:no    Case-sensitive: no
Valexp:
Valmsg:
Help: Please enter the title of the sales representative
Desc:
```

After entering all the information for the slstitle field, press **[GO]** (F1).

```
Field-Name: slsquota                      Data-Type: decimal
Format: $>>, >>>, >>>9                 Extent: 0
Label: Yearly Quota                       Decimals: 0
Column-label:                             Order: 50
Initial:                                  Mandatory: no (not null)
Component of -> View:no Index:no Case-sensitive: no
Valexp: slsquota > 100000 and slsquota < 1000000
Valmsg: Annual sales quota must be > 100,000 and < 1,000,000
Help: Please enter the sales representative's annual sales quota
Desc:
```

Did you notice the Valexp field in this definition? It defines a test, or validation expression, to be performed automatically whenever a user enters a value into the slsquota field. This test makes sure the slsquota is greater than 100,000 and less than 1,000,000. The Valmsg appears on the screen to tell the user the parameters for the information they are entering. Any field can have a validation expression to make sure the user does not enter invalid information.

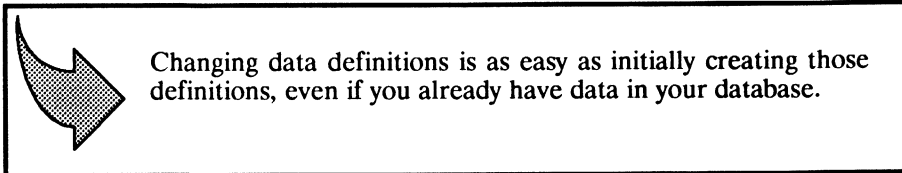
After entering all the information for the slsquota field, press **GO** (F1).

```
Field-Name: hire-date                      Data-Type: date
Format: 99/99/99                          Extent: 0
Label: Date hired                          Decimals: ?
Column-label:                              Order: 60
Initial: TODAY                             Mandatory: no (not null)
Component of -> View:no Index:no Case-sensitive: no
Valexp:
Valmsg:
Help: Please enter the sales representative's date of hire
Desc:
```

The hire-date field definition contains the value TODAY in the initial field. The initial field defines a value that is automatically entered into the field whenever a record is created. The initial value TODAY runs a PROGRESS function that retrieves the current date and places it in the hire-date field. The user can always override an initial value, if necessary, but it can help save time if the appropriate value is chosen.

After entering all the information for the hire-date field, press **GO** (F1). Now you are being prompted for another field name. Since you don't want to enter any more fields, press **END** (F4). PROGRESS displays the Field Editor menu.

If you want to go back and make any changes to the fields you have just defined, choose the Add/Change option from the Display/Change Data Definitions menu and change the appropriate field.



Now that you have described all the parts of the salesrep file, you need to think about how you are going to access that file.

DEFINING AN INDEX FOR THE SALES REP FILE

Think back to the manual system for handling sales rep information that we discussed at the beginning of this chapter.

<p>Sales Rep Data Form</p> <p>Initials:</p> <p>Name:</p> <p>Sales Region:</p> <p>Title:</p> <p>Annual Quota:</p>

If you were going to file these forms in a filing cabinet, what method would you use? You would probably try to file them in a way that made them easy to retrieve. For example, suppose you filed them in order by Sales Region because you thought that would make it easy to generate reports based on regions. There are two drawbacks to that method.

- First, you may not know the region for every sales rep. That would make it very difficult to find information on a particular sales rep.
- Second, even if you did know the region for every sales rep, what would happen when you had two or more sales reps assigned to the same region? When you went looking for a specific sales rep, you would have to leaf through all the forms that had the same region as the sales rep you were looking for.

Instead, you might choose to file them in alphabetical order by the sales rep initials. That way, you can uniquely identify information about a particular sales rep. When you go to find a sales rep data form, you just look for the right set of initials.

It's the same in PROGRESS. Choosing the field that is going to be used most frequently for finding data is called defining an *index*. Let's define an index for the salesrep file.

You are currently at the Field Editor menu and you want to add an index, so choose GoIndex by typing g. This places you in the Index Editor.

```
PROGRESS Data Dictionary          Create New File
MODIFY-SCHEMA  SQL  Database  Admin  Utilities  Reports  Exit
----- Currently Defined Fields -----

NextPage  PrevPage  Add  Modify  Delete  Copy  GoIndex  SwitchFile
Browse  Undo  Exit

----- Total Fields: 0 -----

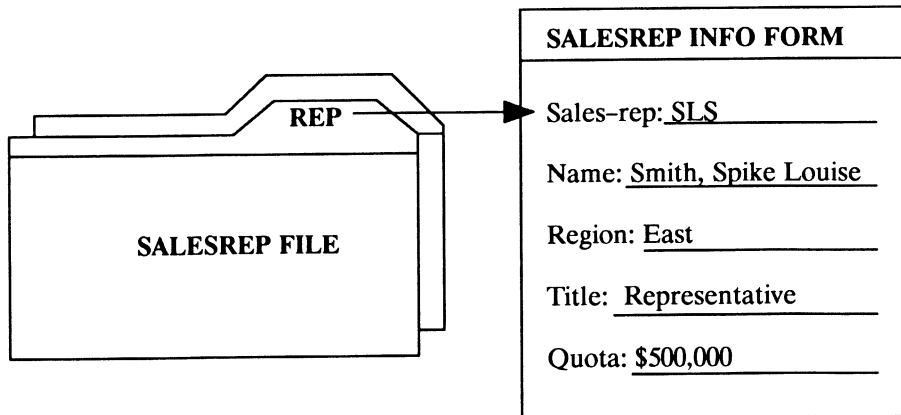
Database: mydrive (PROGRESS)          File: salesrep

Go to the Index Editor.
```

Here is the Index Editor screen:

PROGRESS Data Dictionary		Create New File	
MODIFY-SCHEMA SQL Database Admin Utilities Reports Exit			
Index: default	Primary:	Unique: No	Active: Yes
default	Seq	Field Name	Type Asc Abbr
Next Prev First Last Rename Add Delete ChangePrimary MakeInactive Browse SwitchFile GoField Undo Exit			
Database: mydrive (PROGRESS)		File:salesrep	
Look at the next index of this file.			

You want to add an index, so type **a** for Add. Now you need to choose a name for the index so you can easily refer to that index later when you want to access information in the salesrep file.



Type **rep** as the index name and press **RETURN**. PROGRESS displays default values in the unique and active fields and, for now, skips over the primary field. By deciding what values to put in these fields, you give the index certain characteristics. Consider the following:

- Is this the primary index? When you are processing records (the data) in a file, PROGRESS automatically accesses the records according to the primary index for that file, unless you specify otherwise. Therefore, you usually choose as a primary index a field (or set of fields) that you will use most frequently for processing all the records in a file. When you first name an index, PROGRESS skips over this field and goes directly to the unique field. After you finish supplying information about this index, PROGRESS sets the default value to yes if it is the first index you've defined, and no for each subsequent index. In order to change the primary index, you must select `ChangePrimary` from the Index Editor menu.

In the case of the salesrep file, we have established that you are most often going to want to use the sales rep's initials to look up information in the salesrep file. Since this is the first index for the file, PROGRESS automatically makes it a primary index.

- Is it unique? That is, can two or more sales reps have the same initials? We will assume that this will not happen. Type a `y` and press `RETURN`. This will make the rep index unique.
- Do you want the index to be active? When an index is active, PROGRESS uses that index to access information in the file. On the other hand, when an index is not active, PROGRESS ignores it.

Deactivating indexes speeds the dump and load processes. It is also useful when you want to create a unique index but your database contains non-unique data. You can deactivate the index when you define it and then activate it using a PROGRESS utility after you have made the data unique. You may also want to deactivate an index if it is used only occasionally. This saves the overhead of maintaining the index whenever a value is changed.

We want the rep index to be active, so press `RETURN` to accept the default answer, yes.

After you press `RETURN`, you see this next screen.

PROGRESS Data Dictionary Create New File

MODIFY-SCHEMA SQL Database Admin Utilities Reports Exit

Adding Index "rep"

Num	Field Name	Type	Asc?
	hire-date	date	
	sales-rep	char	
	slsname	char	
	slsquota	dec	
	slsrgn	char	
	slstitle	char	

Use the cursor-motion keys to navigate, or type field name and hilite will move.

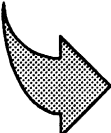
As you select fields to be key components, they are moved above the dividing line. Use [RETURN] to select fields. To se asc/desc, use +/- keys.

To unselect fields, move hilite above dividing line and [RETURN]. When done, press [F1].

Database: mydrive (PROGRESS) File: salesrep

Use [RETURN] to add component, [F1] to save, [F4] to undo.

Now you must list the fields that make up the index. That is, which field values do you want to use as a means of looking up records in the file? In the case of the sales rep file, you are just going to use the sales-rep field (which contains the initials of a sales rep) as the means of looking up information in the salesrep file. Move the cursor down one space to **sales-rep** and press RETURN.



You can combine multiple fields into a single index.

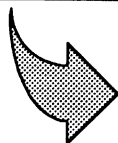
Now, do you want to refer to the sales rep records in ascending (from a-z) or descending (from z-a) order by the sales-rep field? The default is ascending and it's probably a good idea to leave it that way: The sales rep field will be accessed in ascending order alphabetically by initials (unless you specify otherwise elsewhere in the application). Simply press RETURN to accept the default value, yes, in the ascending column.

The highlighted bar is now positioned on to the next line, indicating that you can include another field in the index if you want. Defining an index that is made up of multiple fields is useful in some situations. For example, if you know that you can uniquely access a record by knowing not just one field value but two field values, you may want to index the file based on both of those fields. Since you only need the initials to access a record, press **[GO]**.

You now see a box that asks if you want users to be able to abbreviate the initials of a sales rep. You probably do not, since the first one or two letters of a sales rep's initials are not very meaningful. Press **[RETURN]** to accept the default value, no.

The next box asks if you want to deactivate your index. There are no duplicate values in the index, so you can press **[RETURN]** to accept the default, no.

You are now back in the Index Editor menu and the rep index is now listed under current indexes. Do you want to define any more indexes for the salesrep file? That is, are there any other fields, besides the one containing the sales reps' initials, that you may use to access the file? If there were, you could add those indexes just as you defined the rep index.



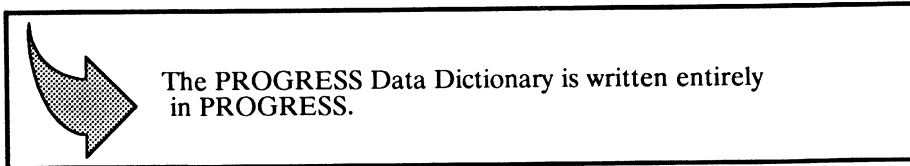
With PROGRESS, you have the power to define multiple indexes for a single file, ensuring rapid access to the records in that file.

For now, let's stick with one index. Press **e** to exit the Index Editor and then then press **[RETURN]** to apply your changes and return to the Data Dictionary Main Menu.

```
PROGRESS Data Dictionary                               Main Menu
Modify-Schema  SQL  Database  Admin  Utilities  Reports  Exit

Database: mydrive (PROGRESS)                           File: salesrep
Edit Files, Fields and Indexes
```

Just one interesting point before you begin to write applications. The Data Dictionary is written in PROGRESS. That is, just as the Test Drive application in Chapter 3 is a PROGRESS application, the Data Dictionary is also a PROGRESS application.

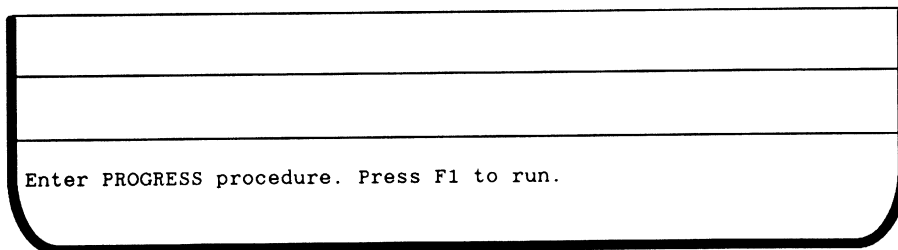


The fact that the Dictionary is a PROGRESS application is yet another indication of the power of PROGRESS as an application development tool.

Now that you have used the Data Dictionary to define the fields and index for the salerep file, you can begin to write procedures that use that file. You use the PROGRESS editor to write your procedures. To get to the PROGRESS editor, press **END** (F4).

WRITING PROCEDURES TO DO THE APPLICATION'S WORK

Now that you are back in the editor, the bottom of your screen looks like this:



The area between the two horizontal lines is the *edit area*, which is where your cursor is now. As you will see, the edit area expands as you type lines for a procedure.

PROGRESS stores the procedures you write as regular ASCII operating system files. That means you could examine them with operating system commands or even edit them with a different editor.

EDITING KEYS

While you were running the Data Dictionary, you used several function keys. Now that you're actually writing your own application, there are some other keys you

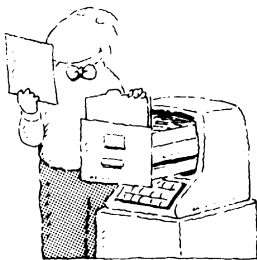
can use. They are the edit keys shown on your key reference card. (You'll find this reference card at the back of this book, if you haven't already.)

The keys vary from terminal to terminal. Remember, if your keyboard has PF keys, use those keys instead of the function keys shown on the reference card. If your keyboard has neither PF keys nor function keys, use the control keys shown on the reference card.

Later, when you explore the PROGRESS documentation that accompanies the Test Drive, you'll see the complete list of PROGRESS keys in Chapter 2 of the *Programming Handbook*.

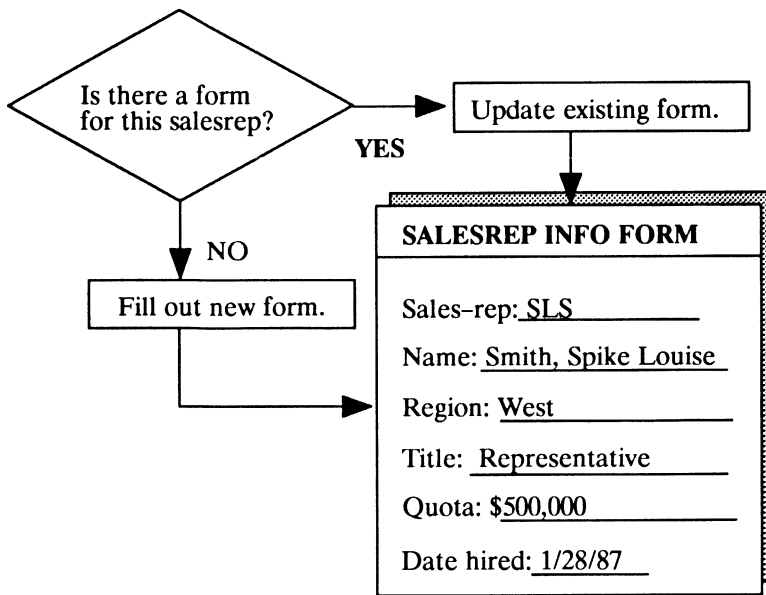
Now, on to writing procedures!

USING PROGRESS TO ADD AND CHANGE DATA



Before you defined the data for the salesrep file, we asked you to imagine how you might describe that data if you weren't using a computer. In the same way, before writing a procedure you should think about the task you want to do and then how you would do it in a manual system.

For example, to add any kind of information to a paper file, what would you do? First, you'd check to make sure that a form didn't already exist for the sales rep. If one didn't exist, you'd get a blank form, fill it out, and put that form into the appropriate place in the file.



You can write a PROGRESS procedure to do this process. We'll just follow the same steps shown in the diagram.

Imagine a user sitting at a computer terminal. That user wants to add a new sales rep to the database.

First, you find out from the user what the sales rep's initials are. You use the PROMPT-FOR statement to get information from the user. Type this statement into the edit area:

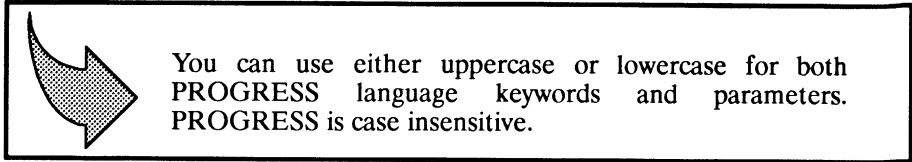
```

PROMPT-FOR sales-rep.

Enter PROGRESS procedure. Press F1 to run.
  
```

In this book, procedures are shown in both uppercase and lowercase. Words that are part of the PROGRESS language (PROMPT-FOR in this example) are shown in uppercase. Arguments, or words you supply, are shown in lowercase (sales-rep in this example). Normally you would supply arguments specific to your

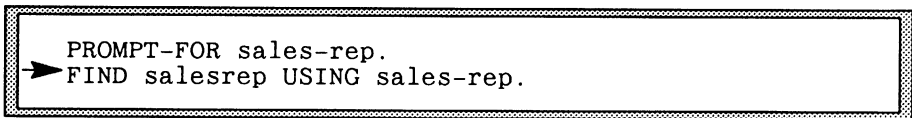
application. However, for the sake of the application being demonstrated here, please type the arguments as shown in the examples.



Also, notice the statement that you are using: PROMPT-FOR. Just by looking at this statement, you can tell that this part of the procedure will prompt the user for some information. You will find that all of the PROGRESS statements are English-like.

Press `RETURN` to move the cursor to the next line in the edit area.

Once the procedure has prompted the user for the sales rep's initials and the user has entered those initials, you want to check to see if that sales rep is already in the database. You use the FIND statement to locate records in the salesrep file:



The arrow (➔) you see in this example is used only to highlight a certain line. You do not type an arrow.

A couple of additional points about statements:

- Statements end with a period.
- Statements can take up more than one line. Just be sure to put a period at the end of the entire statement.

This FIND statement uses the value of the sales-rep field (the sales rep's initials) to find the appropriate sales rep record in the salesrep file.

When you use the FIND statement, PROGRESS tries to find a single record from a database file. If it cannot find that record, it displays an error message and does not process the rest of the procedure. In our example, if a record is not available, you want to create one, right? Therefore, you need to tell PROGRESS not to

display an error message even when it can't find a record. Add NO-ERROR to the FIND statement:

```
PROMPT-FOR sales-rep.
➔ FIND salesrep USING sales-rep NO-ERROR.
```

End the FIND statement with a period and press `RETURN` to move the cursor to the next line in the editing area.

Now, let's suppose that the record does not exist in the database. Remember the figure above that showed how you would want this to work in a manual system? You want to create a new record and assign the sales rep's initials to that record as well as supply all the other pertinent information. Type these lines into the edit area:

```
PROMPT-FOR sales-rep.
FIND salesrep USING salesrep NO-ERROR.
➔ IF NOT AVAILABLE salesrep THEN DO.
➔   CREATE sales-rep.
➔   ASSIGN sales-rep.
➔   UPDATE slsname slsrgn slstitle slsquota hire-date
➔         WITH 1 COLUMN 1 DOWN.
➔ END.
```

Note that many of the lines in this procedure are indented. The indentation is purely for convenience. Indentation and formatting have no effect at all on how the procedure runs.

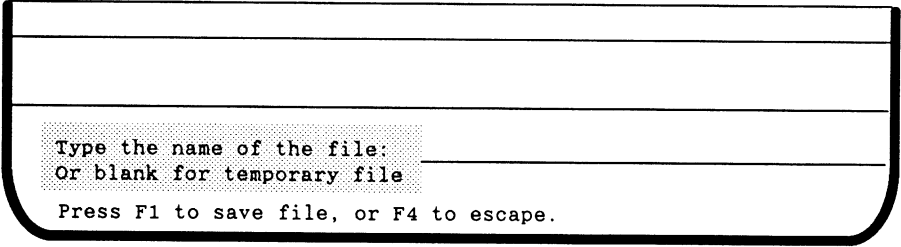
Here, if a record for the sales rep does not exist, PROGRESS creates one and assigns to, or attaches to, that record the sales rep's initials that were supplied in the PROMPT-FOR step. Then the user is allowed to supply the rest of the sales rep information: name, region, title, quota, and date hired.

Now, what if the sales rep record *does* already exist? In that case, you probably want to allow the user to make changes to the information in the record. To do so, add the following lines:

```
PROMPT-FOR sales-rep.  
FIND salesrep USING sales-rep NO-ERROR.  
IF NOT AVAILABLE salesrep THEN DO:  
  CREATE salesrep.  
  ASSIGN sales-rep.  
  UPDATE slsname slsrgn slstitle slsquota hire-date  
    WITH 1 COLUMN 1 DOWN.  
  END.  
➔ ELSE DO:  
➔   NEXT-PROMPT slsname.  
➔   UPDATE salesrep.  
➔ END.
```

This procedure is ready to go. But before you run it, you should save it so it will be available the next time you want to use it. To save the procedure, press **PUT** (F6).

The bottom of your screen looks like this:



Type the name of the file: _____
Or blank for temporary file _____
Press F1 to save file, or F4 to escape.

The procedure name is simply the name of the file in which you want to store the procedure. Let's call this procedure **adchgrep.p** (for add, change sales rep). Type the name of the procedure and press **RETURN**.

The .p file extension is a naming convention that you can use for your PROGRESS procedures. If you give each of your procedure files the .p extension, they will be easier to locate in your directory, and PROGRESS will be able to apply certain defaults when compiling and running your procedures.

Since this is the first time you have written and saved a procedure, you might want to check to be sure a file was really created. That's simple. First, press **CLEAR** (F8) to clear the procedure out of the edit area. Then, if you are using DOS or OS/2, type the following into the edit area:


```
DOS dir.
```

```
OS/2 dir.
```

This is actually a one line procedure that goes out to DOS or OS/2 and performs the directory command. If you are using UNIX, type this into the edit area:

```
UNIX ls.
```

If you are using VMS, type this statement:

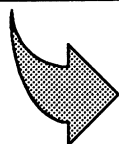
```
VMS dir.
```

If you are using BTOS/CTOS and have the Context Manager installed on your system, type this statement:

```
BTOS List.
```

Now press **[GO]** (F1). (If you are using BTOS/CTOS, press the key labeled GO when the Command List form appears.)

You can see that there is a file called adchgrep.p in your directory. Press **[SPACEBAR]** to return to the PROGRESS editor.



PROGRESS gives you access to the operating system, either for interactive use or to run specific commands from within a procedure.

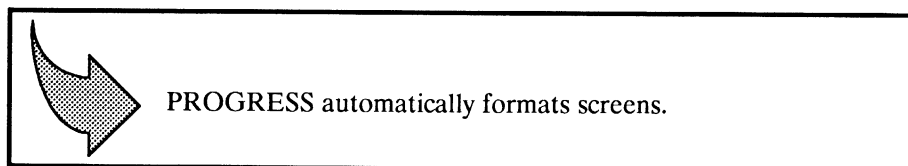
To retrieve the procedure, press **[GET]** (F5). PROGRESS displays the name of the last procedure you saved. Press **[RETURN]** to copy the procedure into the edit area.

You can go ahead and run this procedure. Just press **GO** (F1). PROGRESS then checks the syntax of the adchrep.p procedure, compiles it, and runs it. Here is what PROGRESS displays:

```
Sales Rep: ██████████
      Name:
      Region:
      Title:
Yearly Quota:
      Date hired:
```

Please enter the sales representative's initials

Notice that PROGRESS displays the data in a box and puts the field labels to the left-hand side of each field. The labels are on the left because the procedure used the phrase WITH 1 COLUMN, but the other formatting characteristics are the defaults used by PROGRESS. In fact, it's very easy to let PROGRESS use default formatting rules for your display screens, and it's also easy to change those defaults in your procedures.



Go ahead, type in some initials – try your own. Because there is no sales rep record with your initials in the salesrep file (there are no records in the file, for that matter), PROGRESS creates a new record and lets you supply the rest of the sales rep information. Go ahead and put in some data for the other fields. Use **RETURN** to move from one field to the next.

To tell PROGRESS when you are finished, press **RETURN** or **GO** (F1) after supplying a value for the last field. PROGRESS puts you back in the editor.

Press **GET** (F5) and **RETURN** to retrieve the adchrep.p procedure. Now press **GO** (F1) to run the procedure again, and supply your initials again. This time

there is a sales rep record with your initials. So, the procedure does not create a new record but lets you change the information in the existing record if you want to. Press **[GO]** (F1) when you are done making changes. PROGRESS puts you back in the editor.

Suppose you want to be able to run the procedure again and again without continually being put back into the editor after each run. Press **[GET]** (F5) and **[RETURN]** to retrieve the adchgrep.p procedure.

Try adding the word “REPEAT:” to the top of your procedure and “END.” to the bottom of the procedure. To make these changes, do the following:

1. Your cursor is at the upper left corner of the edit area. Press the up arrow key to move the cursor above the first letter in the first line. Press **[NEW LINE]** (F9) to create a new line at the top of the procedure.
2. Type **REPEAT:**
3. Use the down cursor key to go the bottom of the procedure and type **END.**

Your procedure should look something like the one shown in the following figure. Notice the message statements. These aren't necessary for the procedure to execute, they are simply to help the user understand what is going on as the procedure runs.

```
➔ REPEAT:
  PROMPT-FOR sales-rep.
  FIND salesrep USING sales-rep NO-ERROR.
  IF NOT AVAILABLE salesrep THEN DO:
➔   MESSAGE "New sales rep created: add information.".
    CREATE salesrep.
    ASSIGN sales-rep.
    UPDATE slsname slsrgrn slstitle slsquota hire-date
      WITH 1 COLUMN 1 DOWN.
    END.
  ELSE DO:
➔   MESSAGE "Sales rep already exists: make changes.".
    NEXT-PROMPT slsname.
    UPDATE salesrep.
➔ END.
```

Enter PROGRESS procedure. Press F1 to run.

If you want to add the help messages to your program, use the up and down cursor keys to position the cursor and the **NEW LINE** (F9) key to create a new line.

Before you run the procedure again, press **PUT** (F6) and **RETURN** to save the new version of the procedure. Now press **GO** (F1) to run the procedure. This time, supply a new set of initials.

```
Sales Rep:  SLS
           Name: _____
           Region: _____
           Title: _____
Yearly Quota: _____
Date hired: _____
```

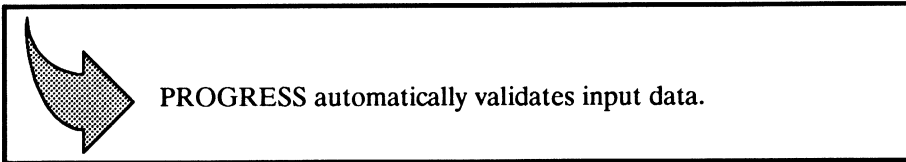
New sales rep created: add information.

Please enter name of sales representative (last name first)

The procedure works the same way as before. It creates a new sales rep record and lets you add more information to that record. But now it also displays a message, telling you that it has created a new record and that you can add information.

Add the rest of the information for sales rep SLS. After supplying a value for the last field, press `[GO]` (F1) to tell PROGRESS you are finished. The last time you did that, PROGRESS returned you to the edit area. But this time, you see the same display again, and you can put in another set of initials. The REPEAT statement tells PROGRESS to run a group of statements again and again until you say you want to stop by pressing `[END]` (F4). Don't press `[END]` (F4) quite yet, though.

On the next sales rep you add, enter some invalid data while running the procedure. Try entering 400 for a sales rep's yearly quota. The terminal beeps. Remember the validation expression? The number 400 does not pass the first half of the test: $400 > 100000$ and $400 < 1000000$.



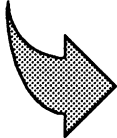
So far, each time you've run the `adchgrep.p` procedure, everything has worked smoothly (with the exception of when you entered some invalid data). What would have happened if there had been a mistake somewhere in the language you used in your procedure? Well, let's try putting a mistake in just to see what happens.

Press `[END]` (F4) to stop running the `adchgrep.p` procedure. PROGRESS returns you to the edit area. Retrieve the `adchgrep.p` procedure using `[GET]` (F5) and replace the word `FIND` with `XIND`. Then move the cursor to any other line in the program. Now press `[GO]` (F1) to run the procedure. When you press `[GO]`, PROGRESS automatically compiles the procedure before running it, checking for any language mistakes. This time through, PROGRESS displays an error message, telling you what is wrong.

```
REPEAT:
  PROMPT-FOR sales-rep.
  XIND salesrep USING sales-rep.
  .
  .
  .

** Unable to understand after -- "XIND". (247)
** Invalid statement.. (254)
Enter PROGRESS procedure. Press F1 to run.
```

PROGRESS not only displays a message that explains the error but also positions the cursor on the line that contains the error.



The PROGRESS syntax-checking editor speeds many tedious and error-prone aspects of application development.

Suppose that, after reading the error message, you still couldn't figure out what was wrong with the procedure. To get information about an error message, just press **HELP** (F2).

Here is what part of the screen looks like when you press **HELP**:

```
      P R O G R E S S  H E L P
-----
1. Recent Messages
2. Any Messages
3. Keyboard
4. Statements
  .
  .
  .

Use CTRL-C anytime to return immediately.
Enter selection or press F4 to exit
```

Choose Option 1, Recent Messages. This option gives you information on the most recent error message you received and then on prior messages in reverse order.

To return to the PROGRESS editor and to your procedure, just press **CTRL-C**.

Now you know how easy it is to write a procedure that can both add new information to a database and change existing database information. How about deleting information from a database?

USING PROGRESS TO REMOVE DATA

Press **CLEAR** (F8) to clear the adchgrep.p procedure from the editor. Then type the following procedure into the edit area. You don't have to type any of the lines that begin with `/*` if you don't want to. All the text contained between `/*` and `*/` symbols are comments that describe what is going on in the procedure. PROGRESS does not process those comments; they are there for your use.

```

/* Create a variable to store the user's answer */
DEFINE VARIABLE answer AS LOGICAL.

/* Use REPEAT so procedure will run repeatedly */
REPEAT:
  /* Ask for the salesrep's initials */
  PROMPT-FOR salesrep.sales-rep.
  /* Find the salesrep record */
  FIND salesrep USING sales-rep.
  /* Display the salesrep information */
  DISPLAY salesrep.
  /* Assume sales rep not deleted */
  answer = NO.
  /* Ask if user wants to delete the information */
  UPDATE "Remove this salesrep?" answer.
  /* If the answer is yes, then... */
  IF answer THEN DO:
    /* Remove the salesrep record from database */
    DELETE salesrep.
    MESSAGE "Sales rep removed".
  END.
  /* If the answer is no, then... */
  ELSE MESSAGE "Sales rep not removed".
END.

```

To save this procedure, press **PUT** (F6), type the name **delrep.p**, and press **RETURN**. Then go ahead and run this procedure by pressing **GO** (F1). The following screen shows what the display looks like:

Sales Rep Name	Region	Title	Yearly Quota
Date hired	answer		

Please enter the sales representative's initials

The display uses the default columnar format. Because all the fields in the sales rep record don't fit on one line, they wrap to the next line, making it difficult to tell one field from another.

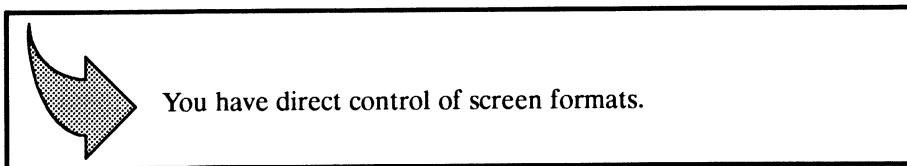
That's easy enough to fix. First, press **END** (F4) to return to the editor, and then press **RECALL** (F7) to redisplay the delrep.p procedure. Now, add these phrases:

```

DEFINE VARIABLE answer AS LOGICAL.

/* Use REPEAT so procedure will run repeatedly */
REPEAT:
  /* Ask for the salesrep's initials */
  PROMPT-FOR salesrep.sales-rep.
  /* Find the salesrep record */
  FIND salesrep USING sales-rep.
  /* Display the salesrep information */
  ► DISPLAY salesrep WITH 1 COLUMN 1 DOWN.
  /* Assume sales rep not deleted */
  answer = NO.
  /* Ask if user wants to delete the information */
  UPDATE "Remove this salesrep?" answer
  ► WITH FRAME answer-frame NO-LABELS.
  /* If the answer is yes, then... */
  IF answer THEN DO:
    /* Remove the salesrep record from database */
    DELETE salesrep.
    MESSAGE "Sales rep removed".
  END.
  /* If the answer is no, then... */
  ELSE MESSAGE "Sales rep not removed".
END.
    
```

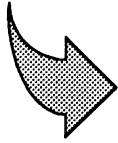
The words "WITH 1 COLUMN 1 DOWN" tell PROGRESS to display the sales rep information in a single column rather than across the screen and to only put a single sales rep record on the screen at any one time.



The words "WITH FRAME answer-frame NO-LABELS" tell PROGRESS to prompt for the user's answer in a different box from the one in which the sales rep information is being displayed and to display no labels for this new frame.

These phrases (WITH 1 COLUMN 1 DOWN, WITH FRAME, and NO-LABELS) describe global formatting characteristics. That is, they describe

how you want to display a whole group of fields. You can also use phrases to describe how you want to display individual fields.



You can use multiple display areas, called frames, on a single screen. You can also display and update information from multiple files in a single frame.

For now, add these phrases (WITH 1 COLUMN 1 DOWN, WITH FRAME, and NO-LABELS) to your procedure as shown in the preceding example. Remember to remove the period from the end of the “UPDATE answer” line since the WITH FRAME phrase is a part of the UPDATE statement. The period then goes at the end of the “WITH FRAME” line.

Save the procedure by pressing (F6) and . Now, let’s run the procedure again. Press (F1).

Here is what the screen display should look like after you enter the initials of an existing sales rep and press .

```
Sales Rep: SLS
      Name: Smith, Spike Louise
      Region: West
      Title: Sales Representative
Yearly Quota: $500,000
      Date hired: 09/02/88

Remove this sales rep? no
```

After you answer **yes** or **no**, press and the procedure prompts you for the next sales rep’s initials.

Press (F4) to return to the editor, and then press (F7) to redisplay the delrep.p procedure.

Let's enhance this display a little with an overlay frame and a frame title:

```

DEFINE VARIABLE answer AS LOGICAL.

/* Use REPEAT so procedure will run repeatedly */
REPEAT:
    /* Ask for the salesrep's initials */
    PROMPT-FOR salesrep.sales-rep.
    /* Find the salesrep record */
    FIND salesrep USING sales-rep.
    /* Display the salesrep information */
    DISPLAY salesrep WITH 1 COLUMN 1 DOWN
    TITLE "Remove a Sales Rep".
    /* Assume sales rep not deleted */
    answer = NO.
    /* Ask if user wants to delete the information */
    UPDATE "Remove this salesrep?" answer
    WITH FRAME answer-frame NO-LABELS
    OVERLAY ROW 7 COLUMN 40.
    /* If the answer is yes, then... */
    IF answer THEN DO:
        /* Remove the salesrep record from database */
        DELETE salesrep.
        MESSAGE "Sales rep removed".
    END.
    /* If the answer is no, then... */
    ELSE MESSAGE "Sales rep not removed".
END.

```

The TITLE phrase in the DISPLAY statement gives a title to the sales rep frame. When the sales rep information is displayed, "Remove a Sales Rep" appears at the top in reverse video or some other attribute defined for your terminal.

The OVERLAY phrase indicates that the answer-frame can appear on top of almost any other frame. The ROW and COLUMN phrases position the answer-frame to specific row and column coordinates on the screen.

When you run this procedure, it first prompts you for a Sales Rep, displays that Sales Rep's information, and then prompts you to press to continue. When you do, the answer-frame is displayed. It overlays the sales rep frame.

Remove a Sales Rep

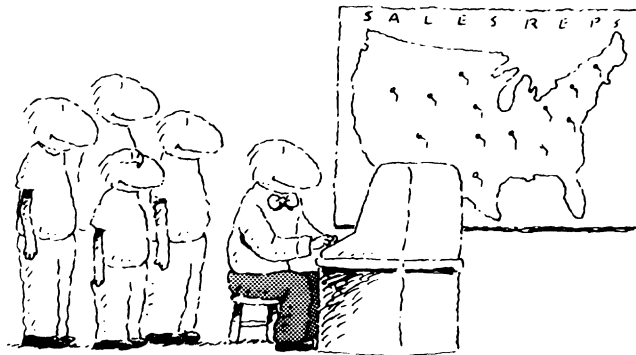
Sales Rep: SLS
Name: Smith, Spike Louise
Region: West
Title: Sales Representative
Yearly Quota: \$500,000
Date hired: 09/02/88

Remove this sales rep? no

You've now written procedures that add sales reps to the database or change existing sales rep records, and that let you delete sales rep records from the database. But what about reports?

PRODUCING REPORTS WITH PROGRESS

Suppose you want to produce a report that lists, for each sales region, the sales rep responsible for that region and the total sales quota expected for that region.



Type the following procedure into the PROGRESS edit area:

```
FOR EACH salesrep BREAK BY slsrgn BY slsname:
  DISPLAY slsname slsrgn slsquota (TOTAL BY slsrgn)
    WITH TITLE "Sales Quotas by Region".
END.
```

In order to see how this works, you're going to have to put some sales rep information into your database. No problem — just run the `adchgrep.p` procedure that you wrote at the beginning of this chapter. Before doing that, take these steps:

1. Save the reporting procedure you just entered in the edit area:
 - Press `PUT` (F6), type the name of the procedure, `showreps.p`, and press `RETURN`.
2. Press `CLEAR` (F8) to clear the edit area.
3. Retrieve the `adchgrep.p` procedure.
 - Press `GET` (F5), type the name of the procedure, `adchgrep.p`, and press `RETURN`.

Press `GO` (F1) to run the `adchgrep.p` procedure and add several sales reps to your database. Then press `END` (F4) to return to the editor.

Now that you have some data in the database, go ahead and run the `showreps.p` procedure. Press `GET` (F5) and retrieve the procedure. Then run it by pressing `GO` (F1). The following figure shows what your screen display should look like (of course, your actual data will be different).

Sales Quotas by Region		
Name	Region	Yearly Quota
Wilson, Robert J.	Central	\$400,000
		\$400,000 TOTAL
Rice, Chris M.	East	\$110,000
Jackson, Betty C.	East	\$125,000
		\$235,000 TOTAL
Garcia, Jerome A.	West	\$400,000
Barlow, John S.	West	\$666,000
		\$1,066,000 TOTAL
		\$1,701,000 TOTAL

Now you know one of the ways to create reports with PROGRESS — by writing a procedure in the PROGRESS 4GL. As an alternative, you can create reports with PROGRESS FAST TRACK. In Chapter 4, you will use the menu-driven features of PROGRESS FAST TRACK to create another report.

You now have three procedures: `adchgrep.p`, `delrep.p`, and `showreps.p`. So far we have run these procedures from the PROGRESS editor. It would be much more convenient if you had a menu from which you could choose to run these procedures.

Press `END` (F4) to return to the editor.

CREATING MENUS WITH PROGRESS

You create menus the same way you just accomplished the other application tasks of adding, changing, deleting, and displaying data: you write a PROGRESS procedure. Of course, you can also create menus with PROGRESS FAST TRACK, as you'll see in Chapter 4.

Type the following procedure into the PROGRESS edit area:

```

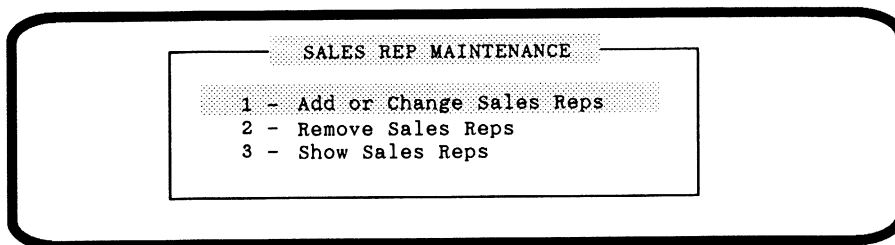
DEFINE VARIABLE proglst AS CHARACTER EXTENT 3 INITIAL [
  "adchgrep.p", /*Add or Change Sales Reps*/
  "delrep.p", /*Remove Sales Reps*/
  "showreps.p" /*Show Sales Reps*/
].

FORM
  WITH FRAME f-cmd ROW 7 CENTERED NO-LABELS ATTR-SPACE
    TITLE " SALES REP MAINTENANCE ".

REPEAT:
  STATUS DEFAULT "Press [" + KBLABEL("END-ERROR") +
    "]" to exit.".
  DISPLAY SKIP(1)
  " 1 - Add or Change Sales Reps" @ proglst[1] SKIP
  " 2 - Remove Sales Reps " @ proglst[2] SKIP
  " 3 - Show Sales Reps " @ proglst[3] SKIP(1)
  WITH FRAME f-cmd.
  CHOOSE FIELD proglst AUTO-RETURN WITH FRAME f-cmd.
  STATUS DEFAULT.
  HIDE ALL.
  RUN VALUE(
    proglst[ INTEGER( SUBSTRING(FRAME-VALUE,2,1) ) ] ).
  HIDE ALL.
END.

```

Press **PUT** (F6) and store this procedure under the name **repmenu.p**. Press **GO** (F1) to run the procedure.



Depending on which menu item you choose, the menu procedure runs a different procedure. If you choose number 1, the menu procedure runs the **adchgrep.p** procedure, and so on. This is one style of menu called a *vertical menu*. There are

other types of menus that you can create with PROGRESS. You'll see another type of menu in Chapter 3.

You wrote the `adchgrep.p`, `delrep.p`, and `showreps.p` procedures as separate procedures and then wrote a fourth procedure, a menu procedure, that could call those three procedures. You also could have written a single procedure that contained the menu and all the statements for the add, change, delete, and reporting operations.

When you're finished working with your menu procedure, press `END` (F4) to return to the editor. You've worked with several procedures now and it may be a little difficult to remember the names that you gave them. That's OK. Just use that one line procedure `UNIX ls`, `DOS or OS/2 dir`, `VMS dir`, or `BTOS List` to look at the contents of your directory.

To do the same thing, you can also press `HELP` (F2), choose option "e" and enter operating system commands interactively. Then if you are using DOS or OS/2, type `EXIT` to return to the Help menu. If you are using UNIX, hold down the `CTRL` key and type `D`. If you are using VMS, type `LOGOUT`. If you are using BTOS/CTOS, type `PROGRESS Exit` and press the key labeled `GO` on your keyboard. When you return to the Help menu, you can press `END` (F4) and you will be back where you were in the editor — even if you were in the middle of changing a procedure.

USING PROGRESS SQL

If your application requires the use of the Structured Query Language (SQL), you'll want to read this section and try out the examples.

ANSI-standard SQL is supplied with your PROGRESS software, and it is integrated into the PROGRESS environment. This means that, when entering SQL statements, you can take advantage of the same high-productivity features you used to enter PROGRESS 4GL statements in this chapter. You can use the PROGRESS full-screen editor to enter and modify your SQL statements, check their syntax, and run them.

In addition, you can define your database files (tables in SQL terminology) with the PROGRESS Data Dictionary, with the SQL data definition language, or both.

Within a single procedure, you can use purely SQL statements, PROGRESS 4GL statements, or mix and match the two languages. Go ahead and enter this SQL query statement. First, press `CLEAR` (F8) if there is a procedure in the editor.


```
➤ SELECT slsname, slsrgrn, slsquota FROM salesrep.
```

Press **GO** (F1) to run the SQL statement. You'll see this screen (except yours will include the information you entered):

<u>Name</u>	<u>Region</u>	<u>Yearly Quota</u>
Wilson, Robert J.	Central	\$400,000
Rice, Chris M.	East	\$110,000
Jackson, Betty C.	East	\$125,000
Garcia, Jerome A.	West	\$400,000
Barlow, John S.	West	\$666,000

Procedure complete. Press space bar to continue.

Here, we're accessing the salesrep file that you created with the PROGRESS Data Dictionary. This SQL statement would also work with a database table created with the SQL data definition language, although the formatting of the data would be different.

But that's no problem — you can use PROGRESS 4GL format and frame phrases to enhance the appearance of any SQL query. Let's give it a try.

First, press **SPACEBAR** to return to the editor. Then press **RECALL** (F7) to redisplay the SQL query statement. Add the following PROGRESS 4GL frame phrase:

```
SELECT slsname, slsrgn, slsquota FROM salesrep  
WITH TITLE "Sales Quotas".
```

Press **GO** (F1) and you'll see that the results of the SQL query now include a frame title:

Sales Quotas		
Name	Region	Yearly Quota
Wilson, Robert J.	Central	\$400,000
Rice, Chris M.	East	\$110,000
Jackson, Betty C.	East	\$125,000
Garcia, Jerome A.	West	\$400,000
Barlow, John S.	West	\$666,000

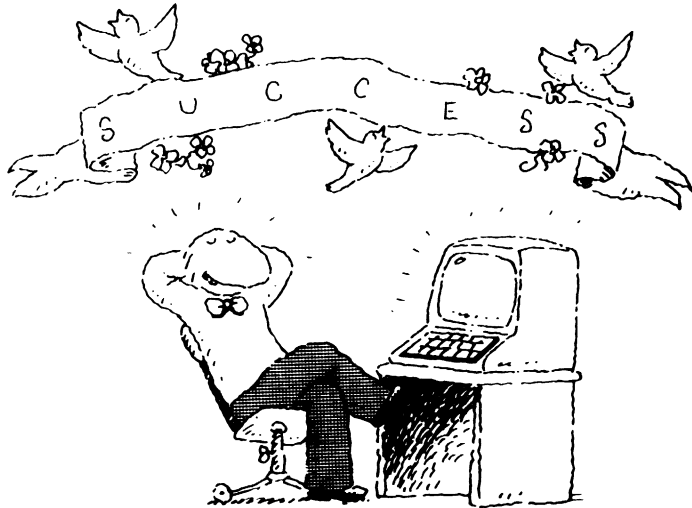
Procedure complete. Press space bar to continue.

Press **SPACEBAR** to return to the editor, and then press **RECALL** (F7) to redisplay the query. To save your SQL query, press **PUT** (F6) and store it under the name **query.p**.

Chapter 5 provides some additional information about PROGRESS SQL.

YOU'VE DONE IT!

You've written your own PROGRESS application and even tried an SQL query. The procedures and menus you've written for your application serve as a strong base for creating ones that are even more sophisticated, like the application procedures you will run in the next chapter.



If you'd like to explore the PROGRESS 4GL further, look through the *PROGRESS Language Tutorial*, *Programming Handbook*, *PROGRESS Language Reference*, and *Pocket Progress*.

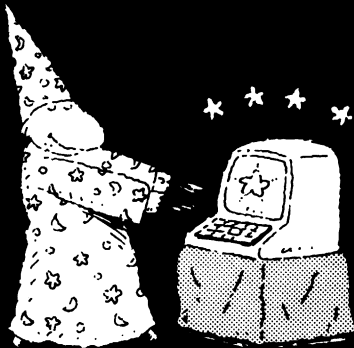
The *PROGRESS Language Tutorial* is designed to teach you the fundamentals of PROGRESS, including the using Data Dictionary as well as the PROGRESS language. Its hands-on approach guides you through the creation of many procedures. If you're interested in more advanced programming techniques, read the *Programming Handbook*. It too contains many example procedures that you can try out. And for complete descriptions and examples of all the statements, functions, phrases, and operators in the PROGRESS 4GL, refer to the *PROGRESS Language Reference*. *Pocket PROGRESS* is a small, easy-to-use reference guide that includes a list of all the PROGRESS startup options and environment variables. *Pocket PROGRESS* also includes a summary of PROGRESS syntax.

When you're ready to exit from PROGRESS and return to your operating system, press **CLEAR** (F8) to clear the edit area, and then type **QUIT** and press **GO** (F1). You'll then see your operating system prompt.

As you explore the PROGRESS 4GL, just remember that you can run the Test Drive only ten times against the same copy of the database. If you run out of sessions, you can either start fresh with a new copy of the empty database, or you can save the data and data definitions you created in this chapter to use in your next ten sessions with the Test Drive. Refer to Appendix A for the details.

CHAPTER 3

THE MAGIC OF A PROGRESS APPLICATION



CHAPTER 3

THE MAGIC OF A PROGRESS APPLICATION

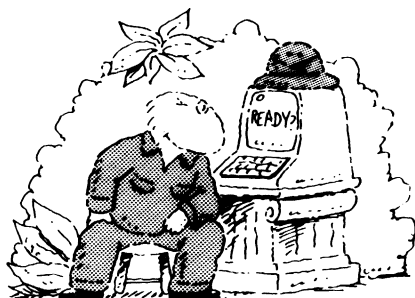
Now that you know a little about PROGRESS, you probably want to know more about the kinds of applications you'll be able to write with it. In this part of your Test Drive, you will run the Test Drive application.

The Test Drive application is not just a simple template of an application. It is a substantial order entry application, one that you might very well use in a business.

As you go through this chapter, keep in mind that there are undoubtedly several ways in which the Test Drive application could be enhanced or extended. The objective of the Test Drive application is to show you the features and capabilities of the PROGRESS Application Development System rather than to show you everything an order entry application can do.

The PROGRESS Application Development System consists of the PROGRESS fourth-generation language (4GL), relational database management system, and PROGRESS FAST TRACK. You've already had some experience with the PROGRESS 4GL and the Data Dictionary when you built your own application in the last chapter. When you use PROGRESS FAST TRACK in Chapter 4, you'll see how this productivity tool speeds the process of creating menus, forms, and reports — the essential building blocks of an application. In this chapter, you'll see one example of the many types of applications you can build with the PROGRESS Application Development System.

Let's go!



ABOUT THE TEST DRIVE APPLICATION

The Test Drive application is an order entry system for a company called All Around Sports. All Around Sports is a sporting goods distributor supplying merchandise to sporting goods retailers.

You saw how PROGRESS works as an application development engine when you built your own application in Chapter 2. In this part of your Test Drive, you are going to run a PROGRESS application. You won't be seeing the application development side of PROGRESS, just the side that lets you run PROGRESS applications.

STARTING THE TEST DRIVE APPLICATION

Before starting the Test Drive, you must install PROGRESS and the Test Drive application according to the instructions in the *Installation Notes* included with your Test Drive package, if you haven't installed them already.

To start the Test Drive, you need to:

- *Log in to your system*

If you are using DOS or OS/2, you probably just need to turn on your computer. If you are using UNIX, VMS, or BTOS/CTOS, type your userid and password, if necessary.

Whether you are running UNIX, DOS, OS/2, VMS, or BTOS/CTOS, be sure you are in the directory from which you want to run the Test Drive application. This directory should be a directory other than the one in which the Test Drive software is installed.

- *Make a copy of the Test Drive database*

Since the Test Drive application has already been written and you are simply going to run it, the first thing you need to do is to get a copy of the Test Drive application's database.

Refer to the following table to find the command for your operating system. Type that command at your operating system prompt.

Operating System	To Copy the Application Database
UNIX, DOS, and OS/2	proddb mydrive2 demo
VMS	PROGRESS/CREATE mydrive2 demo
BTOS/CTOS	PROGRESS Create Database New Database Name mydrive2 Copy From Database Name demo

These commands make a copy of an existing database. In this case, the name of that database is “demo.” The name of your copy of the database is “mydrive2.”

After you type the command, press **RETURN** if you are using UNIX, DOS, OS/2 or VMS. (If you are using a PC, use **ENTER** or **↵** when you see **RETURN** in this book.) If you are using BTOS/CTOS, press the key labeled GO on your keyboard.

Now that you have your own copy of the demo database, you can do whatever you like during the Test Drive without worrying about modifying the original Test Drive database.

- *Start the Test Drive application*

Once you have made a copy of the application database, you can start running the application procedures. As part of that, you will also start PROGRESS.

To start PROGRESS and the Test Drive application, refer to the following table. Type the command for your operating system at the system prompt.

Operating System	To Start the Test Drive Application
UNIX, DOS, and OS/2	drive mydrive2
VMS	@\$DISK:[DLCTDA]drive mydrive2
BTOS/CTOS	Submit File List [Sys] < Sys > drive.sub [Parameters] mydrive2 [Force Expansion?] [Show Expansion?]

If you are using UNIX, DOS, OS/2, or VMS, press **RETURN**. If you are using BTOS/CTOS, press the key labeled GO on your keyboard.

A SPECIAL NOTE FOR TEST DRIVE USERS OUTSIDE THE U.S.

When you start PROGRESS, there are several different startup options you can use. Each describes a different system characteristic. Especially interesting are the two parameters, -d and -E. You can specify -d dmy to indicate that you want all dates shown in day, month, year format rather than in month, day, year format. In addition, if you use the -E parameter, PROGRESS interprets commas as decimal points and decimal points as commas when displaying decimal numbers. Refer to *Pocket PROGRESS* for information about all the start-up parameters.

To use both of these options, you type:

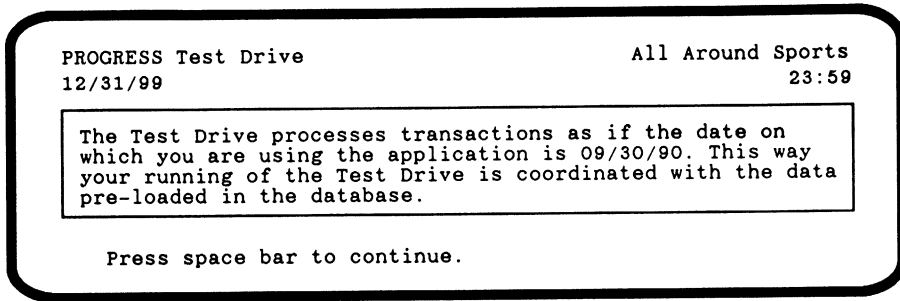
UNIX, DOS or OS/2: drive mydrive2 -d dmy -E

VMS: @\$DISK:[DLCTDA]drive mydrive2
 /DATE_FORMAT=dmy/NUMERIC_FORM
 AT=EÜROPEAN

BTOS/CTOS: Submit
 File List [Sys] < Sys > drive.sub
 [Parameter] mydrive2 -d dmy -E
 [Force Expansion?]
 [Show Expansion?]

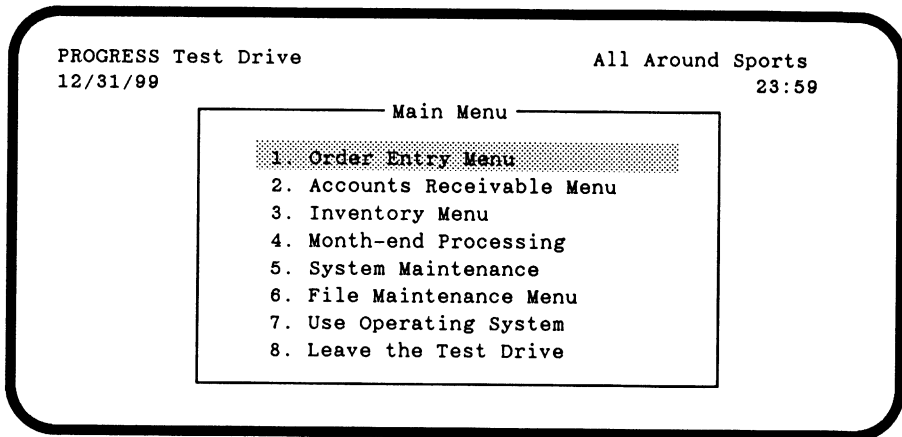
When you start the Test Drive application, you see a welcome banner on your screen. Notice the message at the bottom of the screen: “You may use PROGRESS on this database 9 more times.” Each time you start the Test Drive application, PROGRESS displays a message that tells you how many sessions you have left with the current database. If you run out of sessions, you can make another copy of the demo database and run the Test Drive application ten more times.

Press `SPACEBAR` and you’ll see this screen:



Because the Test Drive application is an order entry application, many parts of the application are dependent upon the dates used for order processing. The data in the application database uses dates between August and September of 1990. Therefore, to assure that the reports and transactions you see look realistic, we assume that the date on which you are using the application is 9/30/90 rather than the current system date.

Go ahead and press `SPACEBAR`. You’ll see the Main Menu of the Test Drive application.



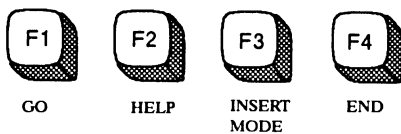
Note the following about this menu:

- The date and time shown in the upper left and right corners of your screen are the current date and time on your system.
- The way in which menu choices are highlighted varies depending on the type of terminal you are using. The highlighting may be shown by reverse video, brightness, or a special color.

The Main Menu is just one of the styles of menus you can create with the PROGRESS 4GL or PROGRESS FAST TRACK. There are other styles which you will see as you run the application.

NOW THAT THE APPLICATION IS RUNNING...

Here are a few keys you'll use while running the Test Drive application:



At the back of this book, you'll find a tear-out reference card that shows many more of the special keys you can use as you go through this chapter. Specifically, you can use the keys that have an Execution Function on the reference card. These keys can be used while running any PROGRESS application.

If your terminal has PF keys, use those keys instead of the ones shown on the reference. For example, use PF1 instead of F1. If your terminal has neither PF keys nor function keys, use the control keys shown on the reference card. For example, **CTRL**-X is the same as F1 on most terminals.

Whenever you need to press a key, you'll see the name of the key followed by the label of the key as it probably appears on your keyboard (some keyboards may be slightly different). For example, we may tell you to press **GO** (F1). This means press the key labeled F1 on your keyboard. This tells PROGRESS to run the highlighted menu option or accept the current form.

Now, let's take a look at the structure of the Test Drive application.

THE APPLICATION STRUCTURE

You can tell a lot about the Test Drive application just by looking at its Main Menu. As mentioned earlier, the Test Drive application is an order processing system for All Around Sports. Here is a brief summary of what the options on the Main Menu allow you to do:

1. Order Entry Menu

Enter orders as they come in, delete orders when they are canceled, update existing orders, and print invoices and reports.

2. Accounts Receivable Menu

Create invoices, enter receipts of cash, print various accounting journals and reports.

3. Inventory Menu

Update inventory quantities or list the items currently in inventory.

4. Month-end Processing

Simulate the closing of a fiscal month and the preparation of journal entries.

5. **System Maintenance Menu**

Control system characteristics, such as the data that appears at the top of every screen in the application (company name, application name, and so on).

6. **File Maintenance**

Add, change, delete, or display customers, items, or sales reps.

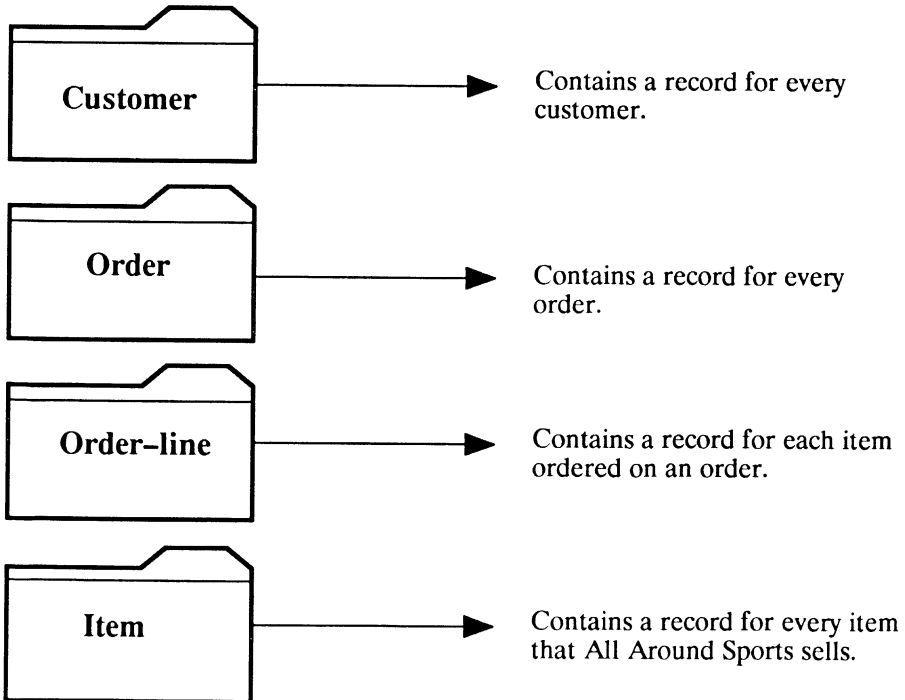
7. **Use Operating System**

Temporarily leave the Test Drive to use UNIX, DOS, OS/2, VMS, or BTOS/CTOS commands.

8. **Leave the Test Drive**

Exit the Test Drive and return to the operating system prompt.

There are several database files set up to store the data used by the application. The four files you'll be seeing as you run the application are:



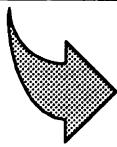
Here's a scenario where each of these files is used:

1. An All Around Sports customer, Dark Alley Bowling, calls All Around Sports to order 20 bowling bags. Spike Smith, an All Around Sports sales representative, answers the phone.
2. Spike checks the customer file to see if Dark Alley Bowling is an existing customer of All Around Sports. If Dark Alley is not a customer, Spike creates a new customer record and stores it in the customer file.
3. Spike creates a new order in the order file and creates a single order-line for the order that specifies a quantity of 20 and indicates the item number for bowling bags. (The order could have multiple order lines, but in this case, there is only one.) The order-line record is stored in the order-line file.
4. After filling out the order form, Spike sends it down to the inventory department at All Around Sports. The inventory manager updates the "allocated" field of the Bowling Bags record in the item file to indicate that 20 bowling bags have been ordered.
5. All Around Sports sends an invoice and the bowling bags to Dark Alley Bowling. Twenty bowling bags are subtracted from the quantity in the "on-hand" field of the Bowling Bags record in the Item file.

Now that you know a bit about what the application does and what the structure of the database is, you can start running the application.

As you run the Test Drive application, keep these points in mind:

- What you are seeing is just one example of the many kinds of applications you can develop with the PROGRESS Application Development System. That is, PROGRESS is *NOT* an order entry system but an application development system with which you can build many different kinds of applications.
- The application is built with PROGRESS alone: no other programming languages were used to write the application you are about to see.



You can build all your applications entirely with PROGRESS; you don't need to use any other language such as C, BASIC, Pascal, or COBOL.

EXPLORING THE TEST DRIVE APPLICATION

To start exploring the Test Drive application, let's first look at what is considered the bare bones of most applications: file maintenance. File maintenance encompasses adding records, changing records, deleting records, and displaying records. All applications do these functions in one way or another.

FILE MAINTENANCE

There are two ways to select an option from the Main Menu:

- Use the up arrow and down arrow keys to move the highlight bar to the option you want and then press `RETURN`.
- Just type the number of the option you want.

Use one of these methods to select option 6, File Maintenance Menu, from the Main Menu. PROGRESS displays this menu:

```
PROGRESS Test Drive                               All Around Sports
12/31/99                                           23:59
```

```
File Maintenance Menu
```

- ```
1. Customer File
2. Item File
3. Sales Rep File
```

```
Menu Path: Main Menu, File Maintenance Menu
```

The File Maintenance Menu is the same style menu as the Main Menu. The options on this menu let you maintain (add, change, delete, or display) information about customers, items, and sales reps.



Choose option 1, Customer File. PROGRESS displays the first customer in the file, Second Skin Scuba:

Customer Maintenance

```

Cust num: 1 Sls rep: SLS Sls reg: West
 Name: Second Skin Scuba Maxcred: 1,500 Unpaid bal: 937.45
 Addr: 79 Farrar Ave Terms: 2% 10/Net30
 Addr 2: Tax num: Disc %: 0
 City: Yuma St: AZ Zip: 85369
 Tel num: (602) 542-0365
 Contact: Ron Ferrante

```

Sales by Month

|          |        |          |        |          |        |
|----------|--------|----------|--------|----------|--------|
| (1)      | (2)    | (3)      | (4)    | (5)      | (6)    |
| 854.15   | 74.34  | 1,462.15 | 144.49 | 1,152.23 | 248.73 |
| (7)      | (8)    | (9)      | (10)   | (11)     | (12)   |
| 1,326.05 | 279.67 | 1,433.07 | 0.00   | 0.00     | 0.00   |

Total YTD Sales 6,974.88

Display the next record

**Next** Prev First Last Seek Query Join View Add Delete Update Output Exit

The line of words (Next, Prev, etc.) at the bottom of the screen is another style of menu, called a *horizontal menu*. The line above the horizontal menu gives you a brief description of the highlighted menu option.

There are two ways to select an option from a horizontal menu:

- Use the right arrow and left arrow keys to move the highlight bar to the option you want and then press `RETURN`.
- Type the first letter of the option you want.

### DISPLAY A RECORD

Go ahead – type N (or n) for Next. PROGRESS displays the next customer in the customer file, Match Point Tennis.

```

Customer Maintenance
Cust num: 2 Sls rep: DKP Sls reg: Central
Name: Match Point Tennis Maxcred: 1,970 Unpaid bal: 77,674.66
Addr: 66 Homer Ave Terms: Net 30
Addr 2: Tax num: Disc %: 0
City: Como St: TX Zip: 75431
Tel num: (817) 498-2801
Contact: Robert Dorr

Sales by Month
(1) (2) (3) (4) (5) (6)
2,126.36 3,071.40 3,150.16 2,546.38 2,126.38 2,415.12

(7) (8) (9) (10) (11) (12)
2,336.37 2,625.13 1,492.78 0.00 0.00 0.00

Total YTD Sales 21,890.06

Display the next record
Next Prev First Last Seek Query Join View Add Delete Update Output Exit

```

## UPDATE A RECORD

Try typing **U** (or **u**) for Update. PROGRESS indicates the data on the screen that you can change. The way update areas appear differs, depending on your terminal type. On some terminals, the fields you can update are underlined, on others they are in reverse video or in a different color.

When you are writing PROGRESS procedures, you have complete control over the colors that are used to display different fields. Not only do you have control, but you can dynamically change those colors while the procedure is running.

As long as you are in update mode, go ahead and make some changes. Change the Maxcred field (the customer's maximum credit allowance) to 1,500. Just press **RETURN** or **TAB** repeatedly to move to that field, and then type in the new maximum credit value.

Let's say that you're not too sure about what kind of data you should put in this field after tabbing over to it. Press **HELP** (F2). This Help menu appears over the Customer Maintenance display:

Enter your choice (You are currently in field customer Max-credit)

```

W What field is this? (Dictionary data)
P Point and shoot a value from this file or lookup-file
B Browse through any file and save a field-value
R Related files (display, save or accumulate data)
F Find and display the entire record
1-9 Put current frame-value into memory 1 to 9
F1-F9 Get a value from memory 1 to 9
C Calculator (numeric)
G General calculator (all functions)
D Date from the calendar
Cursor-right Find next value
Cursor-left Find previous value
Cursor-up .. Find first value
Cursor-down Find last value

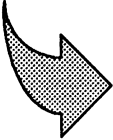
```

| (7)                       | (8)      | (9)      | (10) | (11) | (12) |
|---------------------------|----------|----------|------|------|------|
| 2,336.37                  | 2,625.13 | 1,492.78 | 0.00 | 0.00 | 0.00 |
| Total YTD Sales 21,890.06 |          |          |      |      |      |

Display the next record  
Next Prev First Last Seek Query Join View Add Delete **Update** Output Exit

This sample help facility is, itself, a PROGRESS procedure, named applhelp.p. PROGRESS runs the application's help procedure whenever you press **HELP** (F2).

You can write your own applhelp.p procedure that PROGRESS will run any time your users press **HELP** (F2). Because you can write application-specific help procedures, your applications will be easier for end users to understand and use.



You can include application-specific help in your PROGRESS applications.

Type **w** to select the “What field is this?” option from the Help menu. The applhelp.p procedure displays data definitions for the Max-credit field:

Dictionary data

```

Field name: Max-credit Data-Type: decimal Decimals: 2
Format: -,>>>,>>9 Mandatory: no
Label: Max cred
Col-label: ? Order: 105
Initial: 0 Extent: 0
Valexp: max-credit >= 0 and max-credit <= 9999999
Valmsg: Max credit must be >= 0 and <= 9,999,999
Help: Please enter a credit limit
Desc: Maximum credit

```

---

|          |          |          |          |          |          |
|----------|----------|----------|----------|----------|----------|
| (1)      | (2)      | (3)      | (4)      | (5)      | (6)      |
| 2,126.36 | 3,071.40 | 3,150.16 | 2,546.38 | 2,126.38 | 2,415.12 |
| (7)      | (8)      | (9)      | (10)     | (11)     | (12)     |
| 2,336.37 | 2,625.13 | 1,492.78 | 0.00     | 0.00     | 0.00     |

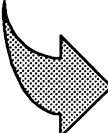
Total YTD Sales 21,890.06

---

Display the next record

Next Prev First Last Seek Query Join View Add Delete Update Output Exit

The Valexp line shows a *validation expression*, or test, that is performed automatically whenever a user enters a value in the Max-credit field. This validation expression makes sure that the maximum credit is greater than or equal to zero and less than or equal to 9,999,999.



Any field in a PROGRESS database can have a validation expression to make sure that the user does not enter invalid data.

The Valmsg field shows a message that PROGRESS displays any time a user enters invalid data. Let's see how this validation expression and message work. Press SPACEBAR to return to the Customer Maintenance screen.

Press CLEAR (F8) to clear the value from the Maxcred field. Now, type **-100** and press RETURN. The validation message appears at the bottom of your screen because -100 is outside the range specified by the validation expression.

Correct the Maxcred value by typing **1500**. To tell PROGRESS to accept the data you've entered, press **GO** (F1).

Now, press **N** to see the next customer record, **Off the Wall**. After looking at that record, press **P** to look at the record for **Match Point Tennis** once again. You can see that PROGRESS has indeed incorporated the change you indicated on that record.

### ADD A RECORD

How about trying to add some information? Press **A** for Add. Now you can add a record for a new customer. Add customer 53, Hurricane Windsurf:

Customer Maintenance

Cust num: 53                      Sls rep: DKP                      Sls reg: Central  
 Name: Hurricane Windsurf      Maxcred: 4,500      Unpaid bal: 0.00  
 Addr: 85 Francis St                      Terms: Net30  
 Addr 2: \_\_\_\_\_                      Tax num: \_\_\_\_\_      Disc %: 0  
 City: Coulee      St: ND      Zip: 58746  
 Tel num: (710)274-5060  
 Contact: Timothy Loff

Sales by Month

|             |             |             |             |             |             |
|-------------|-------------|-------------|-------------|-------------|-------------|
| (1)         | (2)         | (3)         | (4)         | (5)         | (6)         |
| <u>0.00</u> | <u>0.00</u> | <u>0.00</u> | <u>0.00</u> | <u>0.00</u> | <u>0.00</u> |
| (7)         | (8)         | (9)         | (10)        | (11)        | (12)        |
| <u>0.00</u> | <u>0.00</u> | <u>0.00</u> | <u>0.00</u> | <u>0.00</u> | <u>0.00</u> |

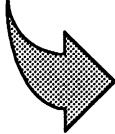
Total YTD Sales 0.00

Next Prev First Last Seek Query Join View Add Delete Update Output Exit  
 Enter data or press F4 to end.

Press **RETURN** after entering each value. That will move the cursor to the next field. After you have entered a value in the Maxcred field, press **GO** (F1) to tell PROGRESS to accept the new customer record.

The Seek choice on the horizontal menu finds a specific record using an index value that you supply. Let's use Seek to display customer number 4, Pedal Power Cycles. Type **s**, and when a blank Customer Maintenance form appears, type **4**. Then press **GO** (F1).





PROGRESS can make sure that you do not accidentally delete a record when other records still exist that are dependent on it, or relate to it. This enforces what is called “referential integrity.”

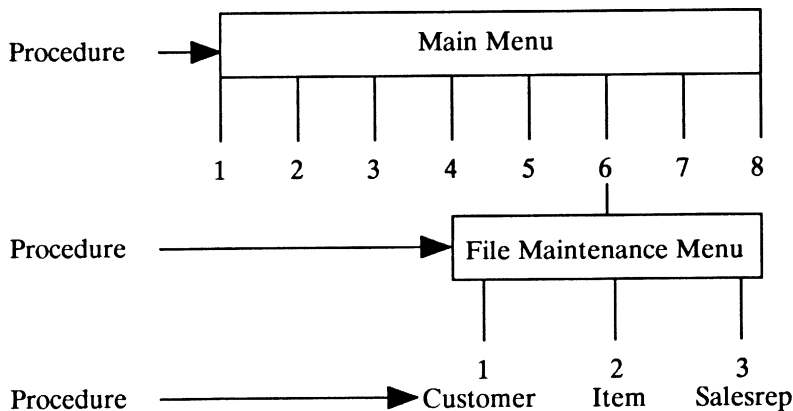
Now you know how to look at, update, add, and delete customer records. You can do the same with sales rep and item records.

That’s all the time we’re going to spend with File Maintenance for now. To get back to the File Maintenance Menu, first press `SPACEBAR` to remove the message and then type `e` for Exit. Once you are at the File Maintenance Menu, press `END` (F4) to return to the Main Menu.

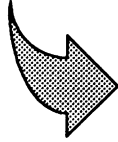
### PROCEDURES AS SUBPROCEDURES

Now you’ve had a look at the Main Menu and the File Maintenance Menu. Are the Main Menu, the File Maintenance Menu and each of the maintenance activities all part of one gigantic program? Definitely not.

The Test Drive application has a structure like this:



Every PROGRESS procedure, whether generated by FAST TRACK or written directly in the PROGRESS 4GL, can run one or more other PROGRESS procedures. For example, the Main Menu of the Test Drive application is displayed by one procedure. When you run that procedure, the Main Menu is displayed. When you choose any of the options from the Main Menu, the Main Menu procedure runs the procedure associated with the option you choose.



PROGRESS procedures can run one or more other procedures written in PROGRESS. You can also call programs written in C from your PROGRESS procedures.

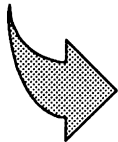
If you are not at the Main Menu, press **END** (F4) repeatedly until you see that menu. Look at the information at the top of your screen: PROGRESS Test Drive, the date, a company name. Now select option 6, File Maintenance. Notice that the File Maintenance Menu has all the same information at the top of the screen.

Now you know that the Main Menu procedure is responsible for running the procedure that displays the File Maintenance Menu. So, how did the File Maintenance Menu access the same information displayed by the Main Menu?

### **PASSING INFORMATION BETWEEN PROCEDURES**

Often, when you have procedures that run other procedures, you want those procedures to be able to pass information to each other, or to at least be able to share a common piece of information.

PROGRESS uses shared variables to store information that is passed between procedures or is shared among procedures. For example, the words “PROGRESS Test Drive” are stored in a shared variable that can be accessed by any PROGRESS procedure that names that variable. The same is true of the company name.



It's easy to pass information from one procedure to another or to share a common piece of data among many procedures.

From the File Maintenance Menu, select option 3, Sales Rep File.



```
PROGRESS Test Drive All Around Sports
12/31/99 Sales Rep Maintenance 23:59

 (A)dd a New Sales Rep
 (C)hange a Sales Rep
 (D)elete a Sales Rep
 (S)how Sales Rep List

Enter selection: █

Enter data or press F4 to end.
```

Notice that the Sales Rep Maintenance menu is a very different style of menu from the one you saw used for either the Main Menu or the File Maintenance Menu. It still works the same way as those other menus: the procedure that displays the menu runs different procedures based on your selection. But why does it look so different?

PROGRESS FAST TRACK and the PROGRESS 4GL let you format your screens, including menus, any way you want. To choose an option from this menu, type the first letter of your selection. Choose A to add a new sales rep.

```
PROGRESS Test Drive All Around Sports
12/31/99 Sales Rep Maintenance 23:59

Sales Rep: █
 Name:
 Region:
 Title:
Yearly Quota:
 Date hired:

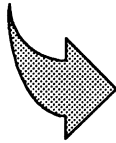
Sales Representative's initials
```

Sales Rep Maintenance is very similar to Customer Maintenance. That is, the Sales Rep Maintenance option lets you add, delete, change, or display information about sales reps. These are the same operations you could do with the Customer Maintenance menu option.

So why does this screen look so much different from the screen you used to work with customer information? For the same reason that the menus you've seen look different.

The procedure that displays the Sales Rep Maintenance screen was written directly in the PROGRESS 4GL. Whenever you write a procedure that displays data, PROGRESS automatically sets up a default display format for that data. You can choose to use that default screen design, or you can modify it to create a screen design that is either slightly different or totally different.

The screen you see here for Sales Rep Maintenance is a default screen designed by PROGRESS. The one used by the Customer Maintenance procedure was created with PROGRESS FAST TRACK. Of course, the Customer Maintenance screen could also be created by writing a procedure with the PROGRESS 4GL. However, as you will see in the next chapter, creating screens with FAST TRACK can be a real time saver.



PROGRESS provides default screen designs that you can modify or replace completely. FAST TRACK's menu-driven interface provides a time-saving alternative to screen design.

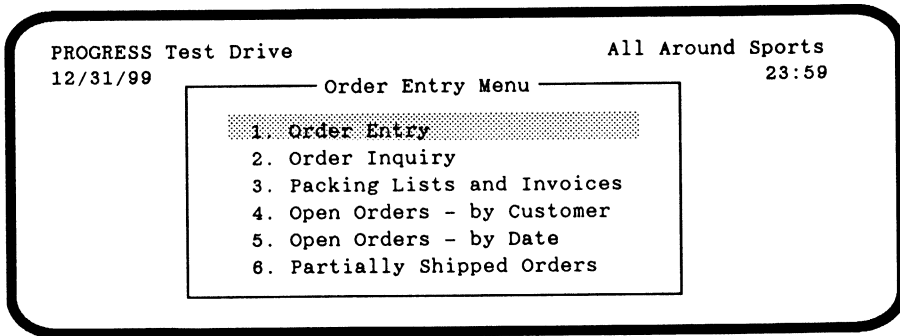
Feel free to explore the File Maintenance Menu options. When you're finished, we'll go on to look at the Order Entry option on the Main Menu.

Press **END** (F4) to return to the File Maintenance Menu. Then press **END** (F4) again to get back to the Main Menu. Once you are back at the Main Menu, choose option 1, Order Entry.

## **ORDER ENTRY**

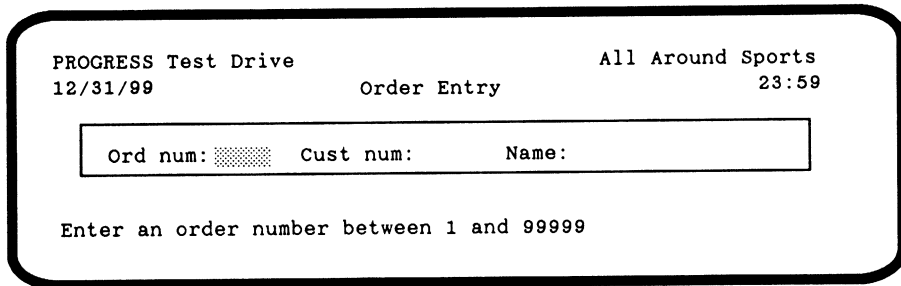
The File Maintenance procedures you just looked at worked with only a single file at a time. That is, the Customer Maintenance procedure manipulated the customer file, the Sales Rep Maintenance procedure manipulated the sales rep file, and so on.

The procedures associated with the options on the Order Entry menu work with multiple files simultaneously. Let's take a look at some of the options on the Order Entry Menu.



This is the same style menu as the Main Menu. You can either use the up and down arrow keys to highlight the option and press **RETURN**, or type the number of the option.

To start, select option 1, Order Entry.



A few minutes ago, you added a new customer, Hurricane Windsurf, to the customer file. Now you can enter their first order. The diagram below shows how various database files are affected by the order.

| Order Form     |                    |
|----------------|--------------------|
| Name           | Hurricane Windsurf |
| Item           | Qty                |
| Swim Goggles   | 10                 |
| Wet Suits      | 2                  |
| Buoyancy Vests | 3                  |

**Customer**

Check to see if Hurricane Windsurf is a customer. If so, display customer data.

**Order**

Add a new order record to the order file.

**Order-line**

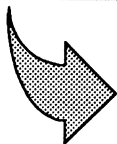
Create new order-line records in the order-line file.

**Item**

Add the quantity of each item ordered to the alloc (allocated) field in each item record.

The first thing to do is to fill out an order form. Let's give this order an order number of 50. Type **50** and press **[RETURN]**.

What was the customer number for Hurricane Windsurf? 55? Try typing **55** and press **[RETURN]**. You get a message telling you that the customer must already exist before you can create an order for that customer.



PROGRESS procedures can validate individual pieces of data and can even perform interfile validation — checking data in one file against data in another file.

Clearly, 55 is not the right number for Hurricane Windsurf. Let's get some help. Press **[HELP]** (F2). From the Help menu, select the option **Point** and shoot a value from this file or lookup-file by typing **p**. This Help option lets you select a value for the current field (in this case, Cust num).

Here's the window that the Help option displays:

PROGRESS Test Drive All Around Sports  
 12/31/99 Order Entry 23:59

Ord num: 50    Cust num: 55    Name:

| Name                 | Cust-num |
|----------------------|----------|
| Batter Up Baseball   | 12       |
| Birdy's Badminton    | 14       |
| Blue Line Hockey     | 13       |
| Buffalo Shuffleboard | 18       |
| Bug in a Rug-by      | 54       |
| Butternut Squash Inc | 8        |
| Chip's Poker         | 29       |
| Dark Alley Bowling   | 51       |
| Fallen Arch Running  | 7        |

Customer must already exist

To find the customer number, you can scroll through the list using the up and down arrow keys, and the page up and page down keys. Or, you can begin typing the customer name. When you do this, the PROGRESS procedure highlights the first customer that begins with the letter or letters you have typed.

Since we know the customer name is Hurricane Windsurf, we can use the name to find the customer number. You need to enter only enough letters to uniquely identify the customer name. Type **Hu**. PROGRESS scrolls the Help display and highlights Hurricane Windsurf, customer number 53. Press **[RETURN]**.

You'll see that the customer number is automatically filled in on the Order Entry screen. Press **[GO]** (F1). The procedure then goes to the customer file, gets the appropriate customer information and displays it, leaving all the order-related information for you to fill in.

|                                 |             |                            |
|---------------------------------|-------------|----------------------------|
| PROGRESS Test Drive<br>12/31/99 | Order Entry | All Around Sports<br>23:59 |
|---------------------------------|-------------|----------------------------|

|             |              |                          |
|-------------|--------------|--------------------------|
| Ord num: 50 | Cust num: 53 | Name: Hurricane Windsurf |
|-------------|--------------|--------------------------|

| Order Data        |                         |
|-------------------|-------------------------|
| Ord date: 9/30/90 | Addr: 85 Francis St     |
| Prom date: / /    | Addr 2:                 |
| Shp date: / /     | City: Coulee            |
| Cust po:          | State: ND Zip: 58748    |
| Terms: Net30      | Misc info: Timothy Loff |
| Ship via:         | Sls rep: DKP            |
| Shp flag:         |                         |

Date of order

At this point, the procedure is working with two files simultaneously: the customer file and the order file. (By the end of this order entry example, you'll see the procedure work with several more files). Not only is it working with more than one file, but you are able to look at, on a single screen, information from more than one file at a time.

Also, you can have multiple display areas, or frames, on a single screen at the same time. Here, the original frame asked you for the order number and the customer number. The second frame lets you update order information.

You can see that screen designs for an application can be very simple (PROGRESS default screen designs), or very elaborate, like the screen shown here in the order entry procedure. Using either PROGRESS FAST TRACK or the PROGRESS 4GL, you can create screens that range from the simple to the complex.

Go ahead and enter the order information shown in this table:

| Field Name                    | Data                                                                                                                                                                                                                                            |
|-------------------------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Ord date                      | Notice that the procedure has already filled in our working date of 9/30/90.                                                                                                                                                                    |
| Prom date                     | This is the shipment date promised to Hurricane Windsurf. Enter a date one month later than the order date.                                                                                                                                     |
| Shp date                      | The date the order is actually shipped. Leave this field blank.                                                                                                                                                                                 |
| Addr, Addr2, City, State, Zip | Information about where to ship the order. The procedure automatically puts the customer's address information here. That way you need change it only if the shipping address for this particular order is different from the customer address. |
| Cust po                       | Hurricane Windsurf's purchase order number. Enter any number you'd like.                                                                                                                                                                        |
| Misc info                     | Any comments you want to make about the order. Notice that the default data in this field is the contact at Hurricane Windsurf.                                                                                                                 |
| Terms                         | The payment terms. The initial value set up for the field in the database is Net30. You can change it if you want to.                                                                                                                           |
| Ship via                      | The method of shipment. Enter Speedy Delivery.                                                                                                                                                                                                  |
| Sls rep                       | The initials of the sales rep responsible for the Hurricane Windsurf account. This field contains the initials DKP, which you assigned to Hurricane Windsurf when you created its customer record.                                              |
| Shp flag                      | An indication of whether or not the order has been shipped. Leave this field blank.                                                                                                                                                             |

Press **GO** (F1) to accept the order data you have supplied. The order entry form appears at the bottom of your screen.

Next Prev Update Add Delete-Line Remove-order Finish

| Line num | Item num | Desc | Qty | Price | Extension |
|----------|----------|------|-----|-------|-----------|
| █        |          |      |     |       |           |

Again, you have multiple frames displayed on the screen: one for the order number, customer number, and name; one for order data; and one for the order

lines. Further, PROGRESS frames can have different characteristics that determine what happens when the screen is full and more information needs to be displayed.

The menu on the bottom of the screen is the same style as the menu you used in the Customer Maintenance procedure. You can highlight the choice you want and press **[RETURN]**, or type the first letter of the choice.

Because this is a new order, you need to add order lines. Type **a** to add order lines. PROGRESS marks the areas you can update. Because this is the first line on the order, enter a **1** for the Line num field. Here is the information for the first order line:

```

Item num 9
Desc Swim Goggles
Qty 10
Price $18

```

After you enter a **1** for the Line num, press **[RETURN]**. Then enter a **9** for the Item num and press **[RETURN]**. Notice that the description of Swim Goggles and the price of \$18 are automatically filled in. The application retrieved that information from the item file in the database. Now enter a **10** for the Qty and press **[RETURN]**. The procedure automatically calculates and displays the cost. You can change the price if you want to and then press **[RETURN]**.

Now type **a** to add another order line. Your screen looks like this:

| Line num | Item num | Desc         | Qty | Price | Extension |
|----------|----------|--------------|-----|-------|-----------|
| 0        | 00000    |              | 0   | 0.00  | 0.00      |
| 1        | 00009    | Swim Goggles | 10  | 18.00 | 180.00    |

The cursor is positioned for you to add another order line. Enter **2** for the line number and press **[RETURN]** to move to the item number field. Now suppose you don't remember the item number you need to enter here. Let's use the "Point and shoot" option on the Help menu to retrieve the item number. Press **[HELP]** (F2) and type **p**. You'll see the following Help display.



|                                 |  |                 |  |                            |  |
|---------------------------------|--|-----------------|--|----------------------------|--|
| PROGRESS Test Drive<br>12/31/99 |  | Order Entry     |  | All Around Sports<br>23:59 |  |
| Ord num: 50                     |  | Cust num: 53    |  | Name: Hurricane Windsurf   |  |
| Ord date: 9/30/90               |  | Desc            |  | Item-num                   |  |
| Prom date: / /                  |  | Baseball Bat    |  | 18                         |  |
| Shp date: / /                   |  | Basketball      |  | 34                         |  |
| Cust po:                        |  | Bowling Bag     |  | 10                         |  |
| Terms: Net30                    |  | Bowling ball    |  | 39                         |  |
| Ship via:                       |  | Bowling Shoes   |  | 38                         |  |
| Shp flag:                       |  | Buoyancy Vest   |  | 7                          |  |
| Date of order                   |  | Croquet ball    |  | 52                         |  |
|                                 |  | Croquet mallets |  | 50                         |  |
|                                 |  | Cycle Helmet    |  | 4                          |  |

Notice that this time when PROGRESS displayed a new frame, it didn't erase any of the old frames. That's because the Help screen is an *overlay frame*. That is, whenever PROGRESS displays this frame, it will place it over any other frames that may appear at the same place on the screen. (You can also tell PROGRESS that you never want a particular frame to be overlaid by another frame. In this case, an overlay frame will not appear on top of that frame.)

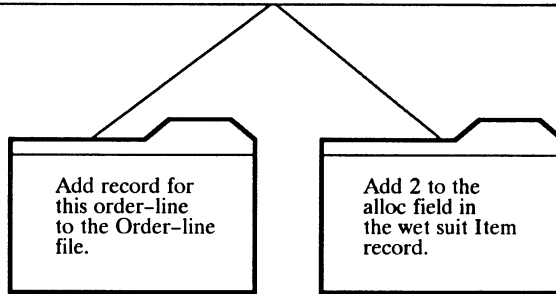
The Help screen contains a list of items and their item numbers. This is the same type of Help screen as the one that listed the customer names and customer numbers. To find the item number you want, you can scroll through the list using the up and down arrow keys, and the page up and page down keys, or you can type the first few characters of the item description until the desired description and number are highlighted.

Use one of these methods to highlight Wet Suits, item number 18. Then press **RETURN**. PROGRESS automatically fills in the Item num, Desc, and Price fields using the selection you made from the Help screen. Now you just need to fill in quantity, for this, press **RETURN** and type 2.

What happens when you enter this order line?

- A new order-line record is created. The quantity of items ordered is stored in the qty field of the new order-line record.
- The record for item number 18, Wet Suits, is found in the item file.
- The number of items ordered (in this case, 2) is added to the alloc (allocated) field in the item record. In other words, those items are set aside so they are ready to go when it is time for delivery.

| <u>Line num</u> | <u>Item num</u> | <u>Desc</u> | <u>Qty</u> | <u>Price</u> | <u>Extension</u> |
|-----------------|-----------------|-------------|------------|--------------|------------------|
| 2               | 00018           | Wet Suit    | 2          | 225.00       | 450.00           |



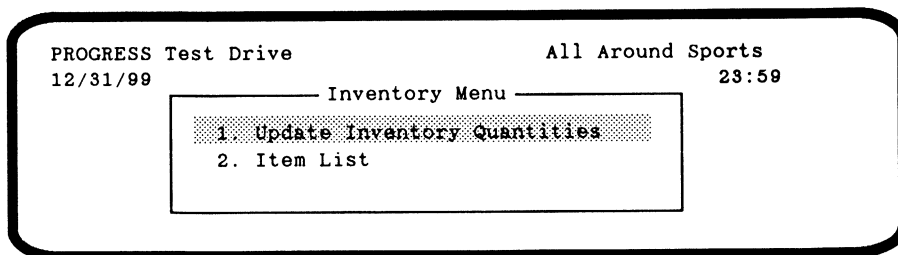
**ORDER-LINE**

**ITEM**

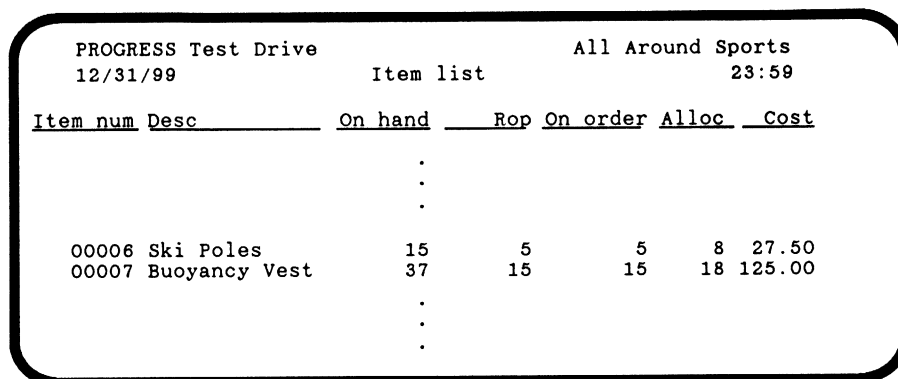
You now have a situation where both the qty field in the order-line file and the alloc field in the item file are being updated together. You wouldn't want one field to be updated and not the other, right? You either want all the processing associated with the order-line to take effect or none of it to take effect; you don't want anything in between.

That is, if the qty field for order-line 2 had 2 in it, but those 2 Wet Suits were not allocated, what would happen come delivery time? Havoc! But this cannot occur with PROGRESS. We'll let you prove it.

The third item on Hurricane Windsurf's order is Buoyancy Vests. Now, let's go see the current allocated amount for buoyancy vests. Press **[GO]** (F1) and then press **[END]** (F4) until you are back at the Main Menu. Once at the Main Menu, select option 3, Inventory Menu.

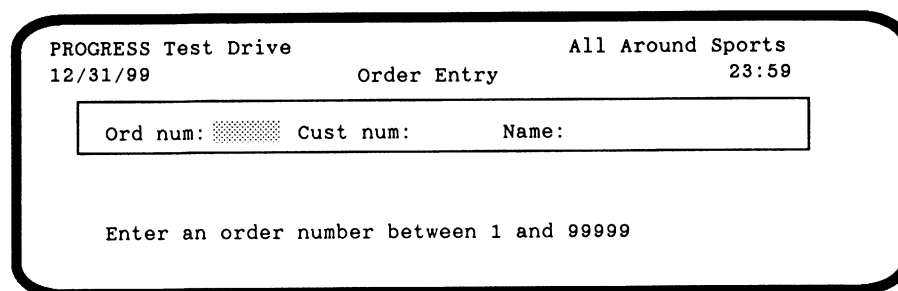


Select option 2 to see a list of items in inventory.



You can see that the number of allocated Buoyancy Vests is 18. Press  three times to see the rest of the report, and then press  again to return to the Inventory Menu. Press  (F4) to return to the Main Menu. Select option 1, Order Entry Menu. Select option 1, Order Entry, from the Order Entry Menu.

Your screen looks like this:



Enter **50** as the order number and press **[RETURN]**. Once the screen fills with the information for order 50, press **[SPACEBAR]**. You can see that order lines 1 and 2 are still there.

| Next     | Prev     | Update       | Add | Delete-Line | Remove-order | Finish |
|----------|----------|--------------|-----|-------------|--------------|--------|
| Line num | Item num | Desc         | Qty | Price       | Extension    |        |
| 1        | 00009    | Swim Goggles | 10  | 18.00       | 180.00       |        |
| 2        | 00018    | Wet Suit     | 2   | 225.00      | 450.00       |        |

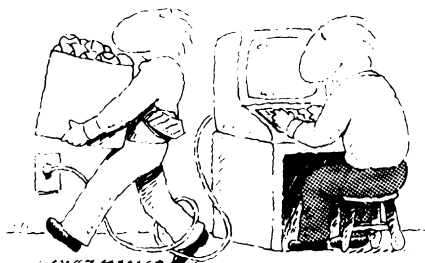
Now type **a** (for Add) to add the next order line.

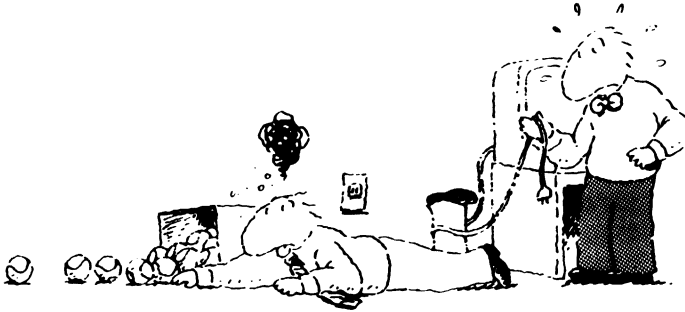
Enter the following, but **DO NOT PRESS [RETURN] WHEN YOU REACH THE PRICE FIELD:**

- 3 for the Line num
- 7 for the Item num
- 3 for the Qty

### WHAT IF YOUR MACHINE GOES DOWN NOW?

What would happen if, for some reason, your system crashed while you were adding this third order line? Say there was a power failure or an operating system error. Let's try to simulate that condition.





### **SIMULATING A SYSTEM FAILURE ON DOS**

If you are using DOS or OS/2, hold down the **CTRL** and **ALT** keys, and press **DEL** all at the same time to reboot your system. If you don't think that's a stern enough test of a system failure (and you have confidence in the ability of your hardware to withstand a power failure), go ahead and pull the plug.

After rebooting, use the DOS and OS/2 CHKDSK command. For example:

```
CHKDSK /F
```

PROGRESS may have created some temporary files which you can remove. The names of these files start with the database name and one of the file extensions .SRT, .PGE, .LBI, or .TRP. When you run the DOS CHKDSK program, it may ask you the following question:

```
Convert lost chains to files (Y/N)?
```

The four temporary files may appear as lost chains. PROGRESS does not need that information so you can answer **no**. But if you think the chains are related to other things on your system, you may want to answer **yes**.

If the mydrive2.lk file exists in your directory, remove that file:

```
del mydrive2.lk
```

Restart PROGRESS as you did before, with the command:

```
drive mydrive2
```

### **SIMULATING A SYSTEM FAILURE ON UNIX**

If you are using UNIX, hold down the **CTRL** key and press the backslash (\) key (or whatever character you are using for stty quit if it's different than backslash).

Once your system comes back up, be sure to get into the same directory you were in, and restart PROGRESS the same way as before:

```
drive mydrive2
```

### **SIMULATING A SYSTEM FAILURE ON VMS**

If you are using VMS, hold down the **CTRL** key and type the letter **y**. When you see the operating system prompt, enter the following command:

```
stop
```

Now restart PROGRESS as you have done before:

```
@$DISK: [DLCTDA]drive mydrive2
```

### **SIMULATING A SYSTEM FAILURE ON BTOS/CTOS**

If you are using BTOS/CTOS, hold down the **ACTION** and **FINISH** keys. When you see the system prompt, restart PROGRESS by typing **Submit** and pressing **RETURN**. The Submit command form appears, as shown below. Type the information shown in bold typeface, and then press **GO**.

```
Submit
File List [Sys]<Sys>drive.sub
[Parameter] mydrive2
[Force Expansion?]
[Show Expansion?]
```

When your system comes back up, restart PROGRESS as you have done before. Type **PROGRESS Single User** and press **RETURN**. When the command form appears, type the information shown in bold typeface:

```
PROGRESS Single User
Database Name mydrive2
```

## HOW PROGRESS RESTORES THE DATABASE

When you restart Test Drive, you see the Test Drive welcome banner followed by these messages:

```
** The last session was abnormally terminated.
** Any incomplete transactions are being backed out.
** Data recovery is complete. You must rerun all
 active transactions.

 Press space bar to continue.
```

Press **SPACEBAR**. The message regarding the working date is displayed. Press **SPACEBAR** again and you'll see the Main Menu.

Remember what the requirement was in the event of a system failure: you wanted both the qty field in the order-line record and the alloc field in the item record to be updated, or neither to be updated. That is, you wanted either all the processing associated with the order line to happen or none of it to happen.

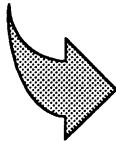
First check the alloc field. Remember that 18 Buoyancy Vests had been allocated before you began entering order line number 3. Select option **3**, Inventory Menu, from the Main Menu. Then select option **2**, Item list, from the Inventory Menu.

You can see that the value in the alloc field of the Buoyancy Vest item record is still 18. No additional Buoyancy Vests have been allocated. Now let's check to see if order line 3 was kept or removed.

Press **END** (F4) repeatedly to return to the Main Menu, then select option 1, Order Entry Menu, from the Main Menu. Select option 1, Order Entry, from the Order Entry Menu. Enter **50** for the order number. Once all the order information is displayed, press **SPACEBAR** to see the order lines. You can see that order line 3 does not exist.

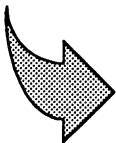
None of the processing associated with order line 3 took effect. That is, the alloc field was unchanged and the order-line record wasn't added to the order-line file. Better than having the order line added to the order-line file but not having the adjusted alloc amount, don't you think?

So, you can see that PROGRESS undid all the processing associated with the order line that was being worked on at the time of the system crash. The procedure that was running the order entry part of the Test Drive application could have specified that more or less work be undone, but this seemed like the best choice for that procedure.



In your application procedures, you can define when and how much work should be undone either in the event of a user error, an application error, or a system failure.

Try this test with any other database system – only be sure to try it with data you don't care about because it just might not work!



In the event of an error or a system failure, PROGRESS provides full, automatic recovery of your database.

And if that's not impressive enough, there are two additional points you should know about how PROGRESS automatically recovers databases:

- PROGRESS undoes whatever work has been done in the transaction. Even if some of the records in the transaction have already been written to disk, PROGRESS restores those records to their pre-transaction condition.



- Not only are files and fields recovered and restored to their pre-transaction condition, but all affected indexes are also restored.

These two points mean that a transaction, or unit of work, is either completed, or completely erased. And no other user or program ever sees a partially updated set of records.

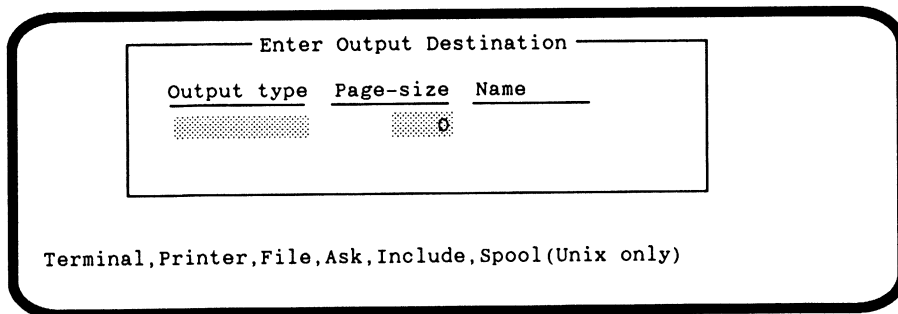
But wait — there's more! Using PROGRESS' Roll Forward Recovery feature gives you another level of protection against the loss of your database. This feature protects your database in the event of media loss.

When Roll Forward Recovery is enabled, PROGRESS records completed transactions in an after-image file, which you store on a separate disk from your database. If the database disk or database files are damaged, you can restore the database by running the after-image file against a backup copy of the database. All the transactions since your last backup are rolled into the backup copy of your database, thus restoring that database to the state it was in at the time of the media damage.

### RUNNING A REPORT

You already ran a simple report when you looked at the item list to check some information about Buoyancy Vests. Let's look at a couple of other reports.

Press **END** (F4) twice to return to the Order Entry Menu. To see an example of one of the reports you can run, choose option **4**, Open orders - by Customer, from the Order Entry menu. You'll see this screen:



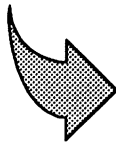
```
Enter Output Destination

Output type Page-size Name

..... 0

Terminal, Printer, File, Ask, Include, Spool (Unix only)
```

This screen prompts you to enter the destination for your report. As you can see from the list at the bottom of the screen, PROGRESS supports many different types of output destinations.



A PROGRESS procedure can send output to and receive input from terminals, operating system files, printers, and other non-terminal devices.

Let's have this report go to your terminal screen. In the Output type field, type t for Terminal and press `[GO]` (F1) to run the report.

PROGRESS displays a message informing you that the output is directed to your terminal. Press `[SPACEBAR]` and you'll see the following report, which lists all the open orders sorted by customer number.

```

PROGRESS Test Drive Page 1 All Around Sports
12/31/99 Open Orders - by Customer 23:59

Number: 1 Name: Second Skin Scuba Credit limit: $1500
Order: 10 Date: 09/27/90 Terms: Net30

Line Item Desc Price Qty Shipped Extension

 1 1 Fins 42.95 56 0 2,405.20
 2 7 Buoyancy Vest 125.00 32 0 4,000.00
 3 24 Snorkel 13.95 76 0 1,060.20
 =====
 Total for order 7,465.40

 Total for customer 7,465.40

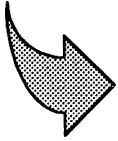
Press space bar to continue.

```

We could have sent the report to the file and to the terminal at the same time. PROGRESS lets you accept input from, or send output to, more than one terminal, file, or device at the same time.

Just by looking at the report, you can see that it is possible to produce very sophisticated reports in PROGRESS. This particular report lists the customer number, the name, the credit limit, the open orders of the customer, each order line on each order, how many items have been shipped, prices, and totals.

Notice the report headers and page numbering, as well as the totals by order and by customer. This report is very detailed.



PROGRESS can handle the most complex reports, including headers, footers, and pre-printed forms.

### PROVIDING INFORMATION TO ANOTHER SYSTEM

Press **[END]** (F4) repeatedly to return to the Main Menu.

Option 4, Month-end Processing, on the Main Menu produces month-end information. It sends accounting transaction information to a file. That file can then be used by another system, such as an accounting package.

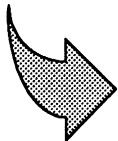
Go ahead and select option 4, Month-end Processing. You'll see this prompt at the bottom of your screen:

```
PROGRESS Test Drive All Around Sports
12/31/99 Month End Processing 23:59

Are you sure that you want to run month-end processing now? yes
```

Press **[RETURN]** to accept the default answer, yes. You'll see a message telling you the name of the file that is being used for output. Press **[SPACEBAR]** to return to the Main Menu.

If you want to look at the contents of the file, choose option 7, Use Operating System. You will see the operating system prompt.



PROGRESS gives you access to the operating system, either for interactive use or to run specific commands from within a procedure.

At the system prompt, enter the command for your operating system, as shown in the following table.

| Operating System | To Display File Contents              |
|------------------|---------------------------------------|
| UNIX *           | more JENTRY                           |
| DOS and OS/2     | type jentry   more                    |
| VMS              | TYPE jentry                           |
| BTOS/CTOS **     | Type<br>File List jentry<br>[Options] |

\* When you enter this UNIX command, be sure to use lowercase and uppercase letters as shown.

\*\* You must have the Context Manager software installed in order to use this BTOS/CTOS command.

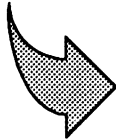
The following screen shows part of the information you'll see when you look at the file's contents. The vertical ellipses indicate missing information:

```

09/30/90 410010101 1,492.78 6 Match Point Tennis
09/30/90 420010101 9.50 6 Match Point Tennis
09/30/90 123450101 -18,548.32Total invoices for the month
09/30/90 100100102 45.00 Match Point Tennis
09/30/90 100100102 89.00 Hoopla Basketball
.
.
.
09/30/90 123450102 -3,689.00Total Receipts for the month
09/30/90 123450103 0.00Total Adjustments for the month

```

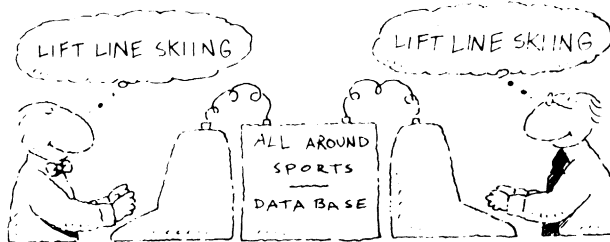
Other systems can use the data in this file as input data. In addition, if you are using other programs that generate data files, PROGRESS can use those data files as input. For example, using the Data Dictionary's Data Exchange facility, you can transfer data between PROGRESS and spreadsheet programs that use the DIF or SYLK format. You can also bring in dBase files for use in PROGRESS.

 PROGRESS applications can exchange data with other applications.

Once you are finished looking at the file, you can return to the Test Drive application. If you are using UNIX, hold down the **CTRL** key and type **d**. If you are using DOS or OS/2, type **EXIT** and press **RETURN**. If you are using VMS, type **LOGOUT** and press **RETURN**. If you are using BTOS/CTOS, type **PROGRESS Exit** and press the key labeled **GO** on your keyboard.

### MULTI-USER APPLICATIONS

If you are using a multi-user operating system, such as UNIX or VMS, you are probably wondering what happens when there are multiple users accessing the same database. Let's find out.



First, leave the Test Drive by selecting option 8 from the Main Menu. To use PROGRESS in a multi-user environment, follow the steps shown below.

**Step 1. Start the PROGRESS server or broker.**

Terminal 1 (UNIX and LANs)

```
$ proserve mydrive2
```

Terminal 1 (VMS)

```
$ PROGRESS/MULTI_USER=START_SERVER mydrive2
```

Terminal 1 (BTOS/CTOS)

```
PROGRESS Server
Database Name mydrive2
[Number of Users]
[Before Image File]
[After Image File]
.
.
.
```

In a multi-user environment, database access is handled by a PROGRESS server (or broker, on shared memory systems). Once the server or broker has been started, multiple users can start PROGRESS.

**Step 2. Start multi-user PROGRESS on two terminals, both logged in to the same directory you have been using all along.**

Terminal 1 (UNIX and LANs)

```
$ mdrive mydrive2
```

Terminal 2 (UNIX and LANs)

```
$ mdrive mydrive2
```

Terminal 1 (VMS)

```
@$DISK:[DLCTDA]mdrive mydrive2
```

Terminal 2 (VMS)

```
@$DISK:[DLCTDA]mdrive mydrive2
```

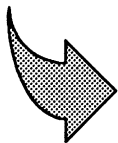
Terminal 1 (BTOS/CTOS)

```
Submit
File List [Sys]<Sys>mdrive.sub
[Parameters] mydrive2
[Force Expansion?]
[Show Expansion?]
```

Terminal 2 (BTOS/CTOS)

```
Submit
File List [Sys]<Sys>mdrive.sub
[Parameters] mydrive2
[Force Expansion?]
[Show Expansion?]
```

Regardless of the operating system you are running on, the Test Drive application behaves the same once you start it. The exact same source code is used for the Test Drive application on all of the computers on which PROGRESS runs.



The PROGRESS 4GL is 100% portable across the more than 200 computers on which PROGRESS runs.

After pressing  twice on both terminals, the Test Drive terminals 1 and 2 should display the Main Menu. Select option **6**, File Maintenance Menu, on both terminals. Both terminals now display the File Maintenance Menu.

One of the reasons a multi-user application environment is very useful is that many users can access the same database. This ensures that all transactions are done with the most accurate, up-to-date information available.

To demonstrate that both Terminal 1 and Terminal 2 are indeed using the same database, do the following:

1. On Terminal 1, select option **1**, Customer File, from the File Maintenance Menu. Type **s** for Seek and then enter a **6** in the Cust num field and press  (F1). Leave Terminal 1 for a moment.
2. Now do exactly the same thing on Terminal 2: type **s**, enter **6**, and press  (F1).

You can see that both Terminal 1 and Terminal 2 are displaying the same information for customer 6, Lift Line Skiing. But that's not really a true test because each terminal could just be using a separate copy of the database. For a true test, do the following:

1. On Terminal 2, press  (F4). After pressing  (F4), you will be back at the File Maintenance Menu.
2. On Terminal 1, type **u** for Update and change the Address for Lift Line Skiing to **1 Main Street**. Press  (F1) to accept the new data.
3. On Terminal 2, select option **1**, Customer Maintenance. Once you see the Customer Maintenance screen, type **s** for Seek and enter **6** in the cust-num field. Now press  (F1). You can see that the information displayed for Lift Line Skiing includes the address change made just moments ago on Terminal 1.

You can see that Terminals 1 and 2 are definitely using the same database.



Great! So you found a database system that allows multiple users to access the same database at the same time, but you need more than that. For example, what would happen in this case:

- Terminal 1 finds and displays customer 6 and Terminal 2 finds and displays customer 6. Terminal 1 updates the customer 6 record but Terminal 2 is still looking at the original version. Now there are two users using the same database, but one has an up-to-date version of a record while one has an old version of the record.
- If Terminal 2 were to update the customer 6 record, that change would overwrite the change just made on Terminal 1.

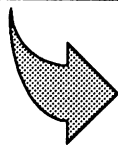
With any traditional programming language and with most other database products, you would be left to solve these multi-user issues yourself. Not so with PROGRESS!

PROGRESS automatically handles data integrity issues arising from multiple users running the same application against the same database. Both Terminal 1 and Terminal 2 have customer number 6 on the screen. Try the following:

1. On Terminal 1, type **u** for Update and change the value of the Maxcred field to **900**. DO NOT PRESS **[GO]** (F1).
2. On Terminal 2, type **u** for Update. PROGRESS displays a message indicating that the record is currently in use by another user. You cannot change the record while Terminal 1 is changing the record.

Here, PROGRESS is applying record locks to ensure that all users are seeing accurate data. Also, because PROGRESS automatically keeps two users from updating the same piece of information at the same time, the integrity of your database is ensured.

3. On Terminal 1, press **[GO]** (F1) to tell PROGRESS to accept the changes you have made.
4. On Terminal 2, notice that the message is gone and that the modified data is displayed and you are now able to update that data.



PROGRESS automatically handles multi-user considerations, detects record contention, and uses appropriate record locks to ensure database integrity.

Of course, if in one or many procedures you want to apply your own record locking rules, you can do so. Simple phrases such as SHARE-LOCK, EXCLUSIVE-LOCK, and NO-LOCK are all you need.

Once you have finished running PROGRESS in multi-user mode on each terminal, return to the Main Menu by pressing **END** (F4) several times and then select option **8**, Leave the Test Drive. Then stop the PROGRESS server or broker that you started earlier.

At the system prompt, type the command for your operating system, as shown in the following table:

| Operating System | To Shut Down the Server or Broker                                                |
|------------------|----------------------------------------------------------------------------------|
| UNIX             | proshut mydrive2                                                                 |
| LANs             | shutdown                                                                         |
| VMS              | PROGRESS/MULTI_USER = SHUTDOWN mydrive2                                          |
| BTOS/CTOS        | PROGRESS Shutdown Server<br>Database Name mydrive2<br>[Server Name]<br>[Options] |

If you are using UNIX, VMS, or BTOS/CTOS, the shutdown command displays user information and the following menu.

```
1 Disconnect a User
2 Unconditional Shutdown
x Exit
```

Enter choice

Type **2** to select Unconditional Shutdown and then press **RETURN**.

## HAVE A LOOK AROUND

This chapter has shown you some of the Test Drive application, but it has not shown you all there is. Feel free to take a look around on your own. The things you've seen so far used the customer, order, order-line, item, and salesrep files. There are several other files used by the Test Drive application: agedar, monthly, shipping, state, and syscontrol.

Remember, if you run out of sessions with the Test Drive application, you can make another copy of the demo database, as explained at the beginning of this chapter, and run the application ten more times.

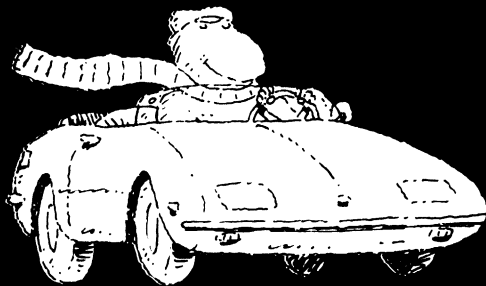
Take the Test Drive application out for a few spins and have a good time!

When you've finished looking around the Test Drive application, go on to Chapter 4. There, you'll learn how to use PROGRESS FAST TRACK to create screens and reports like those you have seen in the Test Drive application.



CHAPTER 4

**ON THE FAST TRACK TO MENUS,  
FORMS, AND REPORTS**





# CHAPTER 4

## ON THE FAST TRACK TO MENUS, FORMS, AND REPORTS

Now that you've had a chance to see a PROGRESS application in action and to write your own application with the PROGRESS language, let's take a look at the productivity tool that makes application development even faster — the PROGRESS FAST TRACK Application Builder.

As you know from writing your own application, PROGRESS automatically formats screens and reports, but also allows you to modify or replace the default designs. As an alternative to using the PROGRESS 4GL to design output, you can use the what-you-see-is-what-you-get (WYSIWYG) features of FAST TRACK.

The image displays three overlapping screenshots of a PROGRESS application. The top-most screenshot is the 'Main Menu', which lists five options: 1. Query database, 2. Update Salesrep File, 3. Create a Salesrep Report, 4. Use Operating System, and 5. Exit. The middle screenshot is the 'Customer Form', which contains input fields for Name, Address, City, St, and Zip. The bottom-most screenshot is a 'CREDIT REPORT' for the 'Eastern Sales Region', showing a table with columns for Name, Unpaid bal, and Max cred, listing various items and their respective values.

**Main Menu**

1. Query database
2. Update Salesrep File
3. Create a Salesrep Report
4. Use Operating System
5. Exit

**Customer Form**

Name: \_\_\_\_\_  
Address: \_\_\_\_\_  
City: \_\_\_\_\_ St: \_\_ Zip: \_\_\_\_\_

**CREDIT REPORT**  
Eastern Sales Region

| Name                | Unpaid bal | Max cred |
|---------------------|------------|----------|
| Off The Wall        | 800.01     | 685      |
| Pedal Power Cycles  | 520.77     | 416      |
| Flying Fat Aerobics | 833.00     | 1,708    |
| Lift Line Skiing    | 1,481.00   | 11,744   |
| Fallen Arch Running | 288.00     | 1,403    |
| Hoopla Basketball   | -1.00      | 1,114    |
| First Down Football | 2,011.00   | 2,187    |
| Batter Up Baseball  | 160.00     | 1,962    |

In addition to creating menus, forms, and reports, FAST TRACK makes it easy to set up the framework of your application and to create query-by-form (QBF) procedures that display database information and accept input from users.

## **BUILDING APPLICATIONS WITH FAST TRACK**

FAST TRACK builds applications from:

- The data definitions you create with the Data Dictionary.
- The procedures you write with the PROGRESS 4GL.
- The menus, forms, reports, and QBF procedures you create with FAST TRACK.

Because FAST TRACK is fully integrated in the PROGRESS environment, you can quickly move between PROGRESS components — the Data Dictionary, the procedure editor, and FAST TRACK.

If you are an experienced programmer, you will see that FAST TRACK speeds up the design and prototyping of crucial portions of your application. You can use FAST TRACK to design the structure and create the building blocks of a brand new application, or you can use FAST TRACK elements to expand an existing application.

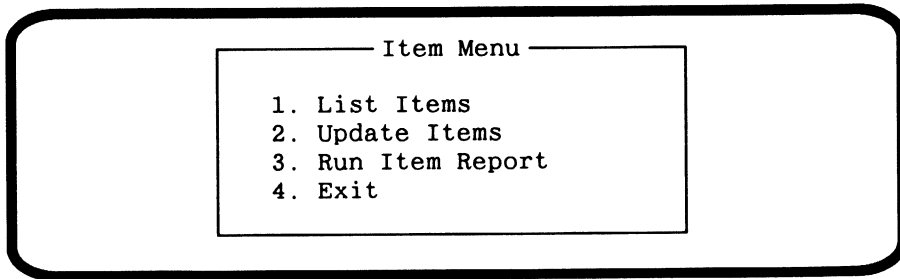
If you have little or no programming experience, you will see that you can quickly learn FAST TRACK. Guided by FAST TRACK's menu-driven interface, you can create special reports, for example, without the help of an application developer, or you can create forms that query the database for special information.

For any user who wants to learn the PROGRESS language, FAST TRACK can serve as a training tool. That is, FAST TRACK generates procedures in the PROGRESS language. You can use these FAST TRACK-generated procedures along with the PROGRESS documentation to learn many PROGRESS programming concepts.

The fact that FAST TRACK generates procedures in the PROGRESS language makes it easy to use those procedures as the basis of a new PROGRESS application or as enhancements to an existing PROGRESS application.

In this chapter, you're going to see many of these FAST TRACK features in action when you create the following menu:





This menu maintains the Item file in the demo database. The Item file contains information about the items that All Around Sports distributes, including the name of each item, its item number, and cost.

After you create the Item Menu, you will assign actions to a couple of the menu options and also design the report to be run by option 3, Run Item Report.

As you go through this chapter, keep in mind that its purpose is to show you the speed and ease with which you can use FAST TRACK to create important parts of an application, rather than to teach you all of FAST TRACK's many capabilities. If you would like to know more about FAST TRACK, refer to the *PROGRESS FAST TRACK Tutorial* and the *PROGRESS FAST TRACK User's Guide*, which are included in your Test Drive package.

Now, let's get on the FAST TRACK to menus, forms, and reports.

## STARTING PROGRESS FAST TRACK

Before starting PROGRESS FAST TRACK, you must install PROGRESS and the Test Drive application according to the instructions in the *Installation Notes* included with your Test Drive package, if you haven't installed them already.

To start FAST TRACK, you need to:

- *Log in to your system*

If you are using DOS or OS/2, you probably just need to turn on your computer. If you are using UNIX, VMS, or BTOS/CTOS, type your userid and password, if necessary.

Whether you are running UNIX, DOS, OS/2, VMS, or BTOS/CTOS, be sure you are in the directory from which you want to run FAST TRACK. This

directory should be a directory other than the one in which the Test Drive software is installed.

- *Make a copy of the Test Drive database*

Since we are going to use the same database you used to run the Test Drive application, the first thing you need to do is to make a copy of that database.

Referring to the following table, find the command for the operating system you are using. Type that command at your operating system prompt.

| Operating System    | To Copy the Application Database                                                       |
|---------------------|----------------------------------------------------------------------------------------|
| UNIX, DOS, and OS/2 | prodb mydrive3 demo                                                                    |
| VMS                 | PROGRESS/CREATE mydrive3 demo                                                          |
| BTOS/CTOS           | PROGRESS Create Database<br>New Database Name mydrive3<br>Copy From Database Name demo |

After you type this command, press `[RETURN]` if you are using UNIX, DOS, OS/2, or VMS. (If you are using a PC, use `[ENTER]` or `[←]` when you see `[RETURN]` in this book.) If you are using BTOS/CTOS, press the key labeled GO on your keyboard.

These commands make a copy of an existing database. In this case, the name of that database is “demo.” The name of your copy of the database is “mydrive3.”

Now that you have your own copy of the demo database, you can do whatever you like while using FAST TRACK without worrying about modifying the original demo database.

- *Start PROGRESS FAST TRACK*

Once you have made a copy of the demo database, you can start FAST TRACK. To do so, you start the Test Drive application and then access FAST TRACK through the procedure editor. To start the Test Drive application, type the command for your operating system at the system prompt:

| Operating System    | To Start the Test Drive Application                                                                             |
|---------------------|-----------------------------------------------------------------------------------------------------------------|
| UNIX, DOS, and OS/2 | drive mydrive3                                                                                                  |
| VMS                 | @\$DISK:[DLCTDA]drive mydrive3                                                                                  |
| BTOS/CTOS           | Submit<br>File List [Sys] < Sys > drive.sub<br>[Parameters] mydrive3<br>[Force Expansion?]<br>[Show Expansion?] |

If you are using UNIX, DOS OS/2,, or VMS, press **RETURN**. If you are using BTOS/CTOS, press the key labeled GO on your keyboard.

When you start the Test Drive application, you see a welcome banner. Notice the message at the bottom of the welcome banner screen: “You may use PROGRESS on this database 9 more times.” Each time you start the Test Drive application with the mydrive3 database, PROGRESS displays a message that tells you how many sessions you have left with this database. If you run out of sessions, you can either start from scratch by making another copy of the demo database, or you can save the data, data definitions, and FAST TRACK objects you create in this chapter and use them in your next ten sessions. Refer to Appendix A for the details.

Press **SPACEBAR** to remove the welcome banner. You’ll see the following screen:

```

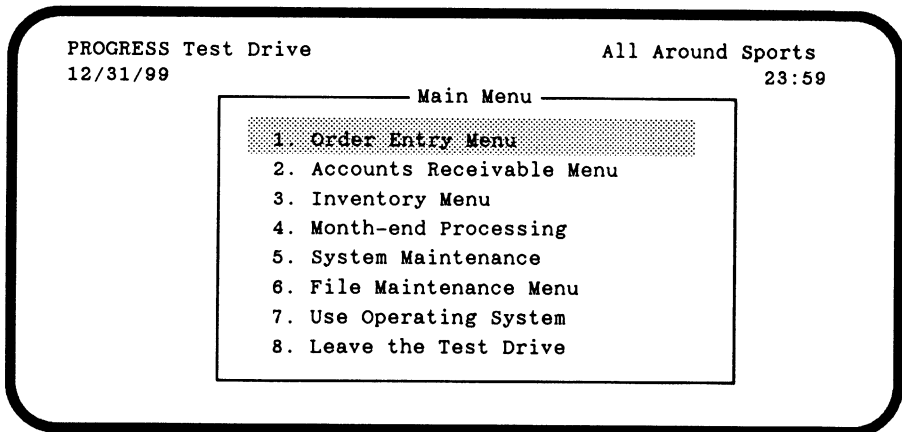
PROGRESS Test Drive All Around Sports
12/31/99 23:59

The Test Drive processes transactions as if the date on
which you are using the application is 09/30/90. This way
your running of the Test Drive is coordinated with the data
pre-loaded in the database.

Press space bar to continue.

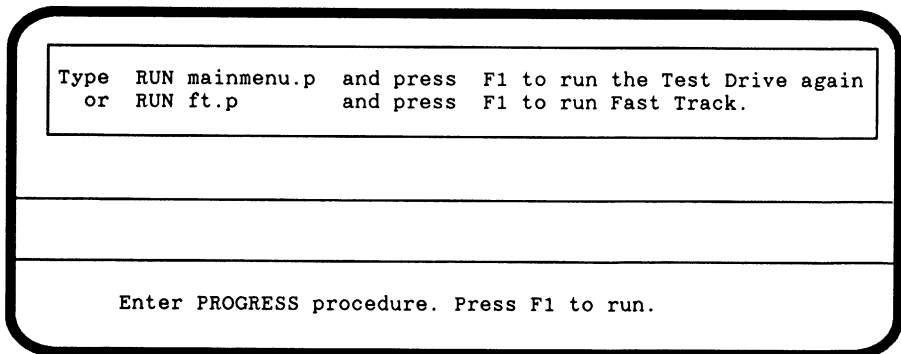
```

Go ahead and press `SPACEBAR` once again. You'll see the Main Menu of the Test Drive application:



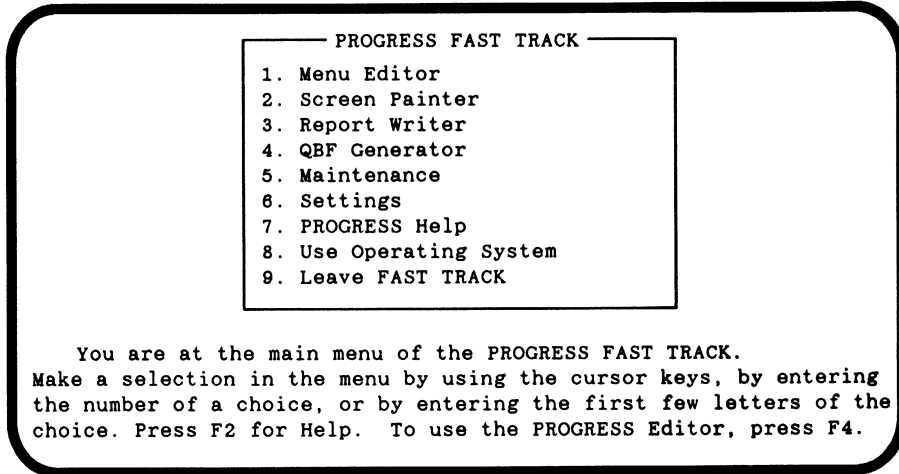
From the Main Menu, we will access the procedure editor. Press the key labeled F4. If you do not have a key labeled F4, you may have a key labeled PF4. Use that key instead. If you have neither the F4 nor the PF4 key, hold down the key labeled CTRL and type e at the same time.

When you press one of these keys, you'll see the following screen.



Because FAST TRACK itself is written in the PROGRESS 4GL, you start it by running the FAST TRACK procedure. Your cursor is between the two lines at the bottom of the screen. Type `run ft.p` and press the key labeled F1 or PF1. If you have neither of these keys, hold down the key labeled CTRL and type x at the same time.

You'll then see the PROGRESS FAST TRACK main menu.



As the main menu shows, FAST TRACK is made up of a number of editors and utilities. Let's take a brief look at these FAST TRACK tools.

### THE MENU EDITOR

Typically, an application has one main menu that accesses other menus, runs reports, runs procedures, or any combination of these. The Menu Editor lets you create the menus and design the overall structure of your application by associating actions with the menu items.

As you build your application, you can view its structure in the form of an outline that shows the relationship of the main menu to the actions you have assigned to it. When you've completed the design of your menu or you want to test it, the Menu Editor generates a PROGRESS procedure to run your application.

You've already had some experience with FAST TRACK menus — when you ran the Test Drive application in the last chapter. Most of that application's menus were created with the FAST TRACK Menu Editor.

### THE SCREEN PAINTER

Most applications use forms to display data or to accept input from users. The Screen Painter allows you to place fields, variables, and text on the screen just as you want them to appear to the user.

When you are satisfied with the layout of your form, you can use it in a PROGRESS procedure or as the basis of a query-by-form (QBF) procedure that you create with FAST TRACK.

### **THE REPORT WRITER**

With the Report Writer, the developer or the end user of an application can create reports using the fields in one or more database files. The Report Writer allows you to automate the production of many standard reports or create one-of-a-kind reports. Without doing any programming, you can produce reports with break groups that provide subtotals, grand totals, and a range of other aggregates.

Once you have laid out a report, you can use it as a building block in your application by instructing the Report Writer to create a PROGRESS procedure.

Most of the reports used in the Test Drive application were created with the FAST TRACK Report Writer.

### **THE QBF GENERATOR**

Through a process called query-by-form (QBF), any user can generate procedures that allow the retrieval and modification of data from one or more database files. You can use the default form the QBF Generator creates, or you can use the Screen Painter to design your own form.

Remember the forms that you used for customer and item file maintenance in the Test Drive application? They were designed with the FAST TRACK Screen Painter, and the procedures to run them were generated with the QBF Generator.

### **MAINTENANCE**

Along with the editors you use to create menus, forms, reports, and QBF procedures, FAST TRACK gives you the tools you need to maintain those objects and to deploy your FAST TRACK application. With the FAST TRACK maintenance tools, you can delete objects from your database, dump and load FAST TRACK data files, deploy your application, and establish security. You can also view an extensive set of reports that allow you to monitor all aspects of the development of your application.

### **AND THERE'S MORE...**

From the FAST TRACK main menu, you also have access to the FAST TRACK key settings, the PROGRESS Help facility, and your operating system.

The FAST TRACK environment provides you with numerous predefined keys that you use to create menus, forms, reports, and QBF procedures. At any time, you can display the current key definitions by simply selecting a menu option.

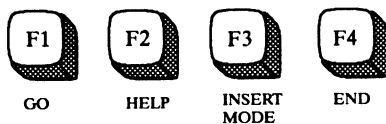
Having access to the PROGRESS Help facility means that you can easily get information about PROGRESS statements, functions, operators, and much more. Through PROGRESS Help, you can also access the Data Dictionary or your operating system, list files in the current directory, and run a PROGRESS procedure. Note that FAST TRACK has an extensive Help facility of its own, which provides information about all FAST TRACK menu options and commands. You'll use the FAST TRACK Help facility later on in this chapter.

You can also access your operating system from the FAST TRACK main menu as well as through the PROGRESS Help facility. This access allows you to temporarily suspend FAST TRACK while you enter operating system commands or run other programs.

There's just one more thing before you begin using FAST TRACK — you need to know a bit about the keys you'll use in this chapter.

### A FEW KEYS YOU'LL USE

As you go through this chapter, you'll use several special keys. Here are a few of those keys:



At the back of this book, there's a tear-out reference card that shows the function keys you can use with PROGRESS and with FAST TRACK. As you go through this chapter, you can refer to the table entitled FAST TRACK Function Keys for a brief explanation of the keys you can use.

If your terminal has PF keys, use those keys instead of the ones shown on the reference card. For example, use PF1 instead of F1. If your terminal has neither PF keys nor function keys, use the control keys shown on the reference card.

Whenever you need to press a key, you'll see the name of the key followed by the label of the key as it probably appears on your keyboard (some keyboards may be

slightly different). For example, we may tell you to press **GO** (F1). This means press the key labeled F1 on your keyboard. This tells FAST TRACK to run the highlighted menu option or accept values entered in a screen.

In some cases, you will use CTRL or ESC key sequences to enter FAST TRACK commands. For example, we may tell you to press **OPTIONS** ( **CTRL** -O ), which means you hold down the key labeled CTRL and type the letter o at the same time. This key sequence displays a menu of FAST TRACK commands. If you are using DOS, however, press and hold the **ALT** key instead of the **CTRL** key.

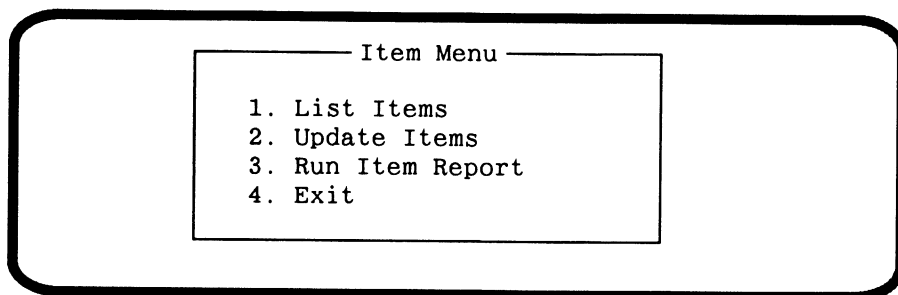
Another key sequence that you will use is **CHOICES** ( **ESC** C ). This means press and release the key labeled ESC and then type the letter c. This key sequence displays a menu of choices for the current FAST TRACK field. If you are using DOS, however, use the **ALT** key instead of **ESC** (press and hold the key labeled ALT and type c at the same time).

And now it's time to experience FAST TRACK for yourself.

## CREATING A FAST TRACK MENU

In Chapter 2, you created a menu by writing a procedure in the PROGRESS 4GL. In this chapter, you will create a menu using FAST TRACK commands, menus, and point-and-pick techniques.

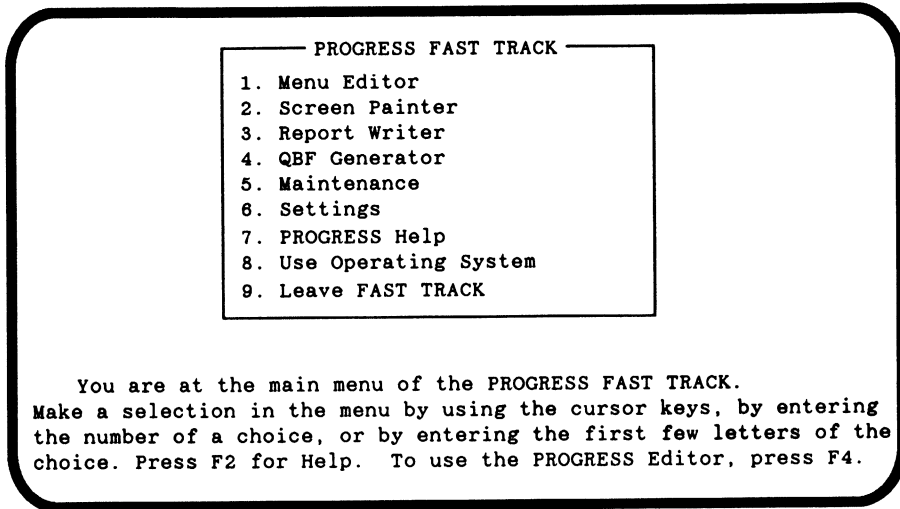
This is the menu you will create:



In addition to creating the menu, you will create the report called by option 3 on the menu and assign a predefined FAST TRACK procedure to option 4. Let's go!



The FAST TRACK main menu is on your screen.



Like the Data Dictionary main menu and others you've used in the preceding chapters, the FAST TRACK main menu is a *vertical menu* — the options are lined up one above the other. There are three ways to select an option from a vertical menu:

- Use the arrow keys to highlight your option and press either **GO** (F1) or **RETURN** .
- Type the first few letters of the menu option.
- If the menu options are numbered, type the number of the option you want.

Since the Menu Editor option is already highlighted, select it by pressing either **GO** (F1) or **RETURN** . You'll see the Menu Editor initialization window on your screen.

Menu Editor

Menu Name:  Menu Title:   
Menu Description:

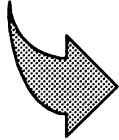
Please enter the name of the menu.  
ESC-C:Choices F1:Go F2:Help F4:Leave

The initialization window prompts you to supply a name for your menu, the title you want to appear on the menu, and for a brief description of the menu. Only the menu name is required, but let's give our new menu both a name and a title. For the menu name, type **Items**.

Any time you make a typing error on a FAST TRACK menu, you can use the left and right arrow keys to position the cursor, then use the  key to delete characters to the left of the cursor or the  key to delete the character under the cursor.

When you have entered the menu name, press  to move the cursor to the Menu Title field. Type **Item Menu**. Press  (F1) to submit the menu name and title. Then you'll see the Menu Editor edit screen.





To make menu creation quick and easy, FAST TRACK automatically numbers your menu options. You can, however, easily override this feature to create unnumbered menus.

For the second menu option, type **Update Items** and press `RETURN`.

Now type **Exit** and press `RETURN`. We forgot the Run Item Report option. No problem. It's easy to go back and insert a menu option.

Press the up arrow key to move the highlight bar back to the Exit line on the menu. Press `INSERT` (F9) to insert a new blank line before the Exit line. Type **Run Item Report** and press `RETURN`.

The text of your menu options is now complete:

```
Item Menu
1. List Items
2. Update Items
3. Run Item Report
4. Exit

Press any alphanumeric key to enter text.
^O:Options F1:Go F2:Help F4:Leave F9:Insert F10>Delete
Menu Name: Items
```

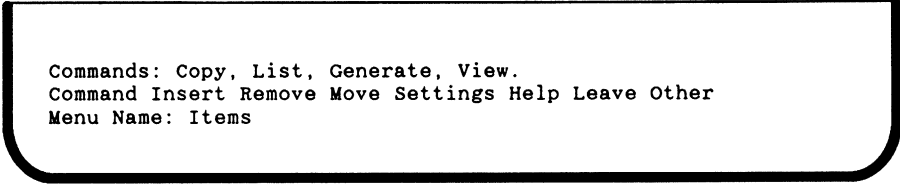
## CHANGING THE CHARACTERISTICS OF A MENU

As you can see from the Item Menu, FAST TRACK provides default characteristics for the menus you create. For example, FAST TRACK

automatically numbers your menu options and positions your menu at the top center of the screen. You can easily change these and many other default characteristics. Let's try it.

If you are using UNIX, or BTOS/CTOS, press **OPTIONS** (**CTRL**-O). (Hold down the key labeled CTRL and type the letter o at the same time.) If you are using VMS, press **ESC**-O (hold down the key labeled ESC and type the letter o at the same time). If you are using DOS, use the ALT key instead of the CTRL key. (Hold down the key labeled ALT and type the letter o at the same time.)

At the bottom of your screen, you'll see a *horizontal menu* of commands on the middle line:



```
Commands: Copy, List, Generate, View.
Command Insert Remove Move Settings Help Leave Other
Menu Name: Items
```

To choose an option from a horizontal menu, use either of the following methods:

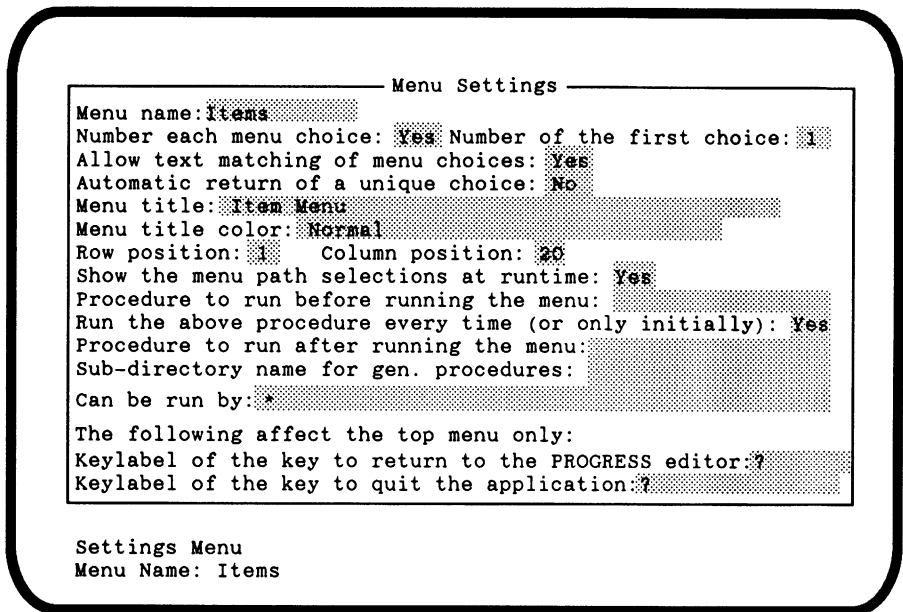
- Use the right and left arrow keys to highlight the option and press **GO** (F1) or **RETURN**.
- Type the first letter of the menu option.

Press the right arrow key to highlight the various options on this horizontal menu. As you do, notice that FAST TRACK displays a brief description of each option as you highlight it.

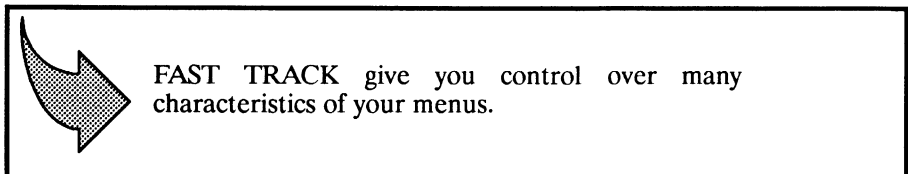
Now let's use this horizontal menu to change the characteristics of the Item Menu.

To show you the options you need to select from horizontal menus, we use an arrow between the options, like this: **SETTINGS**→**MENU**. This means that you use one of the methods described above to select the Settings option, which displays another horizontal menu. From this submenu, you select the Menu option.

Now, go ahead and select **SETTINGS**→**MENU**. Type s for Settings, and then type m for Menu. You'll see the Menu Settings window.

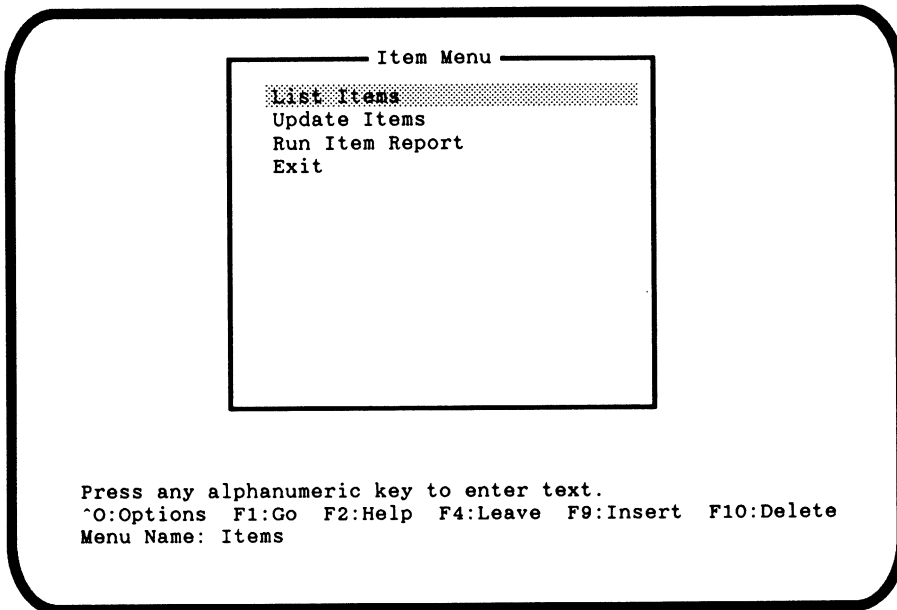


As you can see from looking at the fields in this window, FAST TRACK gives you control over many characteristics of your menus. Along with characteristics that determine the appearance of your screen, you can name PROGRESS procedures that you want to run before and after the menu is run. You can also define security measures.



Let's make the Item Menu an unnumbered menu. Press **RETURN** to move the cursor to the field labeled "Number each menu choice." Type **no**.

Press **GO** (F1) to save the new menu setting and return to the Item Menu. The options on the Item Menu are now unnumbered.



- Since the options on the Item Menu no longer have numbers, you must select an option by highlighting the option and pressing **RETURN**, or by typing the first letter of the option.

Now that you have created the text of your menu and determined the characteristics, you can put it to work.

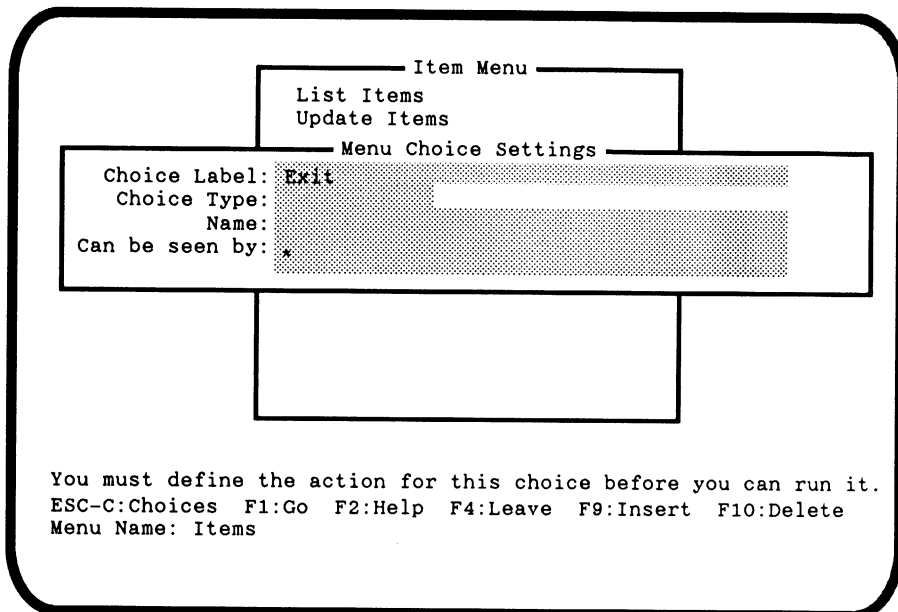
### ASSIGNING ACTIONS TO MENU OPTIONS

By assigning actions to menu options, you determine the structure of your application. That is, you determine what task each option is going to perform, such as:

- Displaying another menu.
- Running a query-by-form procedure created with the FAST TRACK QBF Generator.
- Running a procedure written in the PROGRESS 4GL.
- Running a report designed with the FAST TRACK Report Writer.

Let's begin by assigning an action to the Exit option on your menu. Use the down arrow key to move the highlight bar to the Exit option. Now press **GO** (F1).

The Menu Editor first searches for a defined action to take for the Exit option. Because we haven't yet defined an action for this option, the Menu Editor pops up this window:

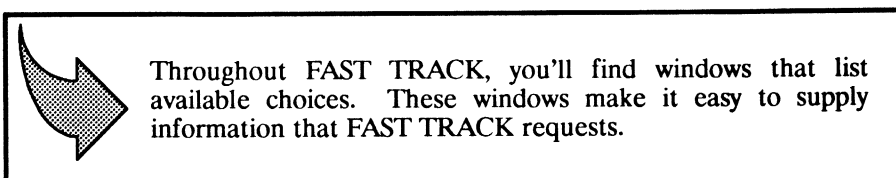
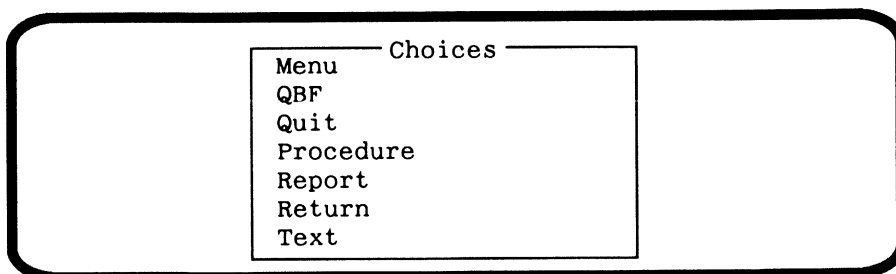


In the Menu Choice Settings window, you enter the type of action you want the menu option to perform, the name of that action, and the users who can select the option. In the Choice Label field, the Menu Editor displays the text of the menu option you're currently working on.

To enter the choice type, press **RETURN** to go to the Choice Type line, then press **CHOICES** (**ESC** C). (Press and release the key labeled ESC and then type c. If you are using DOS or OS/2, remember to press and hold the ALT key whenever you see ESC in this book.)



You'll see a list of the types of actions you can assign to a menu option:

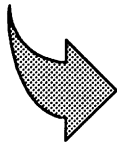


As the Choices window shows, you can have a menu option run another FAST TRACK object (another menu, a QBF procedure, or a report). You can also choose to have a menu option run a procedure that you have written in the PROGRESS language.

In addition, the Choices window includes two predefined PROGRESS procedures that are frequently used as menu options: Quit and Return. Quit exits from the application and returns to the operating system. Return goes back to the previous menu.

Let's assign one of these procedures to the Exit option on our menu. Press the down arrow key until Quit is highlighted and then press **RETURN**. You're now back at the Menu Choice Settings window where you'll see Quit in the Choice Type field.

Because you have selected a predefined procedure, you do not have to enter a name for it. The Can Be Seen By field allows you to indicate the users who can select the menu option. By default, the option is available to all users. This is indicated by the asterisk (\*).



**FAST TRACK** allows you to implement security measures at many levels of your application.

Let's leave the Can Be Seen By field as it is for now. Press **[GO]** (F1) to assign the Quit procedure and return to the Menu Editor edit screen.

To let you know that the Quit procedure has been assigned to the menu option, the Menu Editor displays "User will QUIT now" on the message line near the bottom of the screen. Press **[SPACEBAR]** to remove the message.

Now that you know how easy it is to assign an action to a menu, let's do it again.

#### **CALLING A REPORT FROM A MENU OPTION**

One of the actions you can assign to a menu option is to run a report. You'll assign that action to the Run Item Report option on the Item Menu.

The Exit option is currently highlighted. Press the up arrow key to highlight the Run Item Report option. Press **[GO]** (F1) and the Menu Choices Settings window appears once again:

Item Menu

Menu Choice Settings

Choice Label: Run Item Report  
Choice Type:   
Name:   
Can be seen by: \*

You must define the action for this choice before you can run it.  
ESC-C:Choices F1:Go F2:Help F4:Leave F9:Insert F10>Delete  
Menu Name: Items

Now press **RETURN** to go to the Choice Type line and **CHOICES** ( **ESC** **C** ) to display the Choices window:

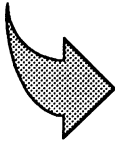
Menu Choices

QBF  
Quit  
Procedure  
Report  
Return  
Text

Use the down arrow key to highlight **Report** and then press **RETURN** . When the Menu Choice Settings window reappears, you'll see **Report** in the Choice Type field.

Press **RETURN** to move the cursor to the Name field. Type **itemrpt** and press **GO** (F1).

FAST TRACK searches for a report named "itemrpt" in the current database. When it finds that there is no report by that name, FAST TRACK takes you right to the Report Writer where you can create the report.



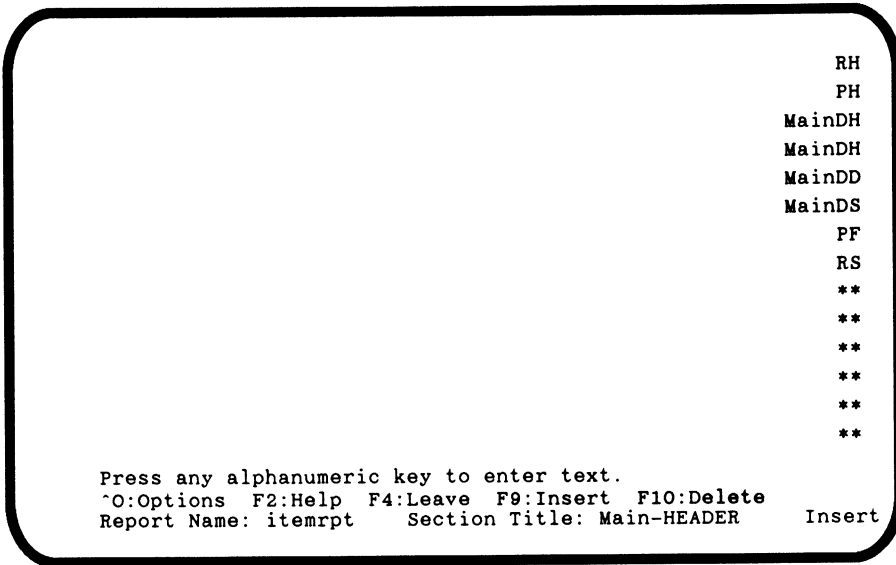
FAST TRACK provides easy access to all parts of the FAST TRACK system as well as to other components in the PROGRESS environment.

## CREATING A REPORT

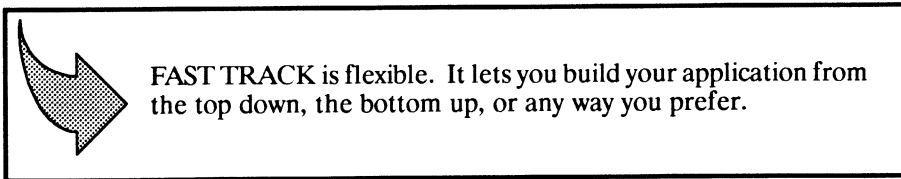
Let's say that you want your report to show information about All Around Sports' inventory, including the quantity on hand, allocated, and on order for each item that the company distributes. Here's what that report might look like:

| Desc            | Item num | On hand | Alloc | On order |
|-----------------|----------|---------|-------|----------|
| Fins            | 00001    | 0       | 80    | 2        |
| Tennis Racquet  | 00002    | 0       | 15    | 7        |
| Sweat Band      | 00003    | 142     | 79    | 65       |
| Cycle Helmet    | 00004    | 141     | 0     | 56       |
| Leotard, Black  | 00005    | 56      | 87    | 27       |
| Ski Poles       | 00006    | 15      | 8     | 5        |
| Buoyancy Vest   | 00007    | 37      | 18    | 15       |
| Runner's Vest   | 00008    | 38      | 0     | 12       |
| Swim Goggles    | 00009    | 0       | 22    | 3        |
| Bowling Bag     | 00010    | 37      | 34    | 9        |
| Lacrosse Stick  | 00011    | 51      | 0     | 15       |
| Rugby Shirt     | 00012    | 61      | 45    | 28       |
| Squash Glove    | 00013    | 0       | 15    | 16       |
| Knee Guards     | 00014    | 86      | 120   | 50       |
| Football Helmet | 00015    | 10      | 3     | 14       |
| Baseball Bat    | 00016    | 98      | 0     | 32       |
| Tennis Balls    | 00017    | 258     | 0     | 74       |
| Wet Suit        | 00018    | 10      | 14    | 4        |

When you entered the name of the report (itemrpt) and FAST TRACK couldn't find a report by that name, it took you right to the Report Writer. Here's the display that's on your screen. It's called the Report Writer edit screen:



Of course, if you decided not to create the item report at this time, you could simply press **END** (F4) to return to the Item Menu. Then whenever you're ready to create the report, you could return to the Report Writer either through the Menu Editor as we have shown you here, or by selecting the Report Writer option on the FAST TRACK main menu.



The edit screen is the area in which you design your report. The items down the right side of the screen outline a default format for your report. Here's a brief description of those items:

|                                                                      |                         |                                                                                      |
|----------------------------------------------------------------------|-------------------------|--------------------------------------------------------------------------------------|
| <b>RH</b>                                                            | <b>Report Header</b>    | A title or banner for your report.                                                   |
| <b>PH</b>                                                            | <b>Page Header</b>      | Information that appears at the top of each page of your report.                     |
| <b>Main DH</b><br><b>Main DH</b><br><b>Main DD</b><br><b>Main DS</b> | <b>Detail Data Area</b> | Column titles, the actual data, and optional summary information, such as subtotals. |
|                                                                      |                         |                                                                                      |
|                                                                      |                         |                                                                                      |
|                                                                      |                         |                                                                                      |
| <b>PF</b>                                                            | <b>Page Footer</b>      | Information that appears at the bottom of each page of your report.                  |
| <b>RS</b>                                                            | <b>Report Summary</b>   | Information that summarizes the data in the entire report, such as grand totals.     |

The first thing you do to create a report is select the database files you want to use in the report. It's possible to create a report using all of the files in your database, but for this report we'll use a single file, the item file.

To select the item file, press **OPTIONS** ( **CTRL-O** ). (If you are using DOS, hold down the **ALT** key instead of **CTRL**; if you are using VMS, hold down the **ESC** key instead of **CTRL**.) Then choose **DEFINE→FILES**. Remember, an arrow between menu options means that you select the first option from the horizontal menu at the bottom of your screen and then select the second option from the submenu that appears.

When you've selected the two menu options, the Report Writer displays this menu:

| Input Files - section: Main |             |       |
|-----------------------------|-------------|-------|
| Db Name                     | Parent Db   |       |
| File Name                   | Parent File | Where |
|                             |             | No    |

ESC-C:Choices F1:Done F2:Help F4:Leave F9:Create F10>Delete  
 Define Files  
 Report Name: itemrpt Section Title: Main-HEADER Insert

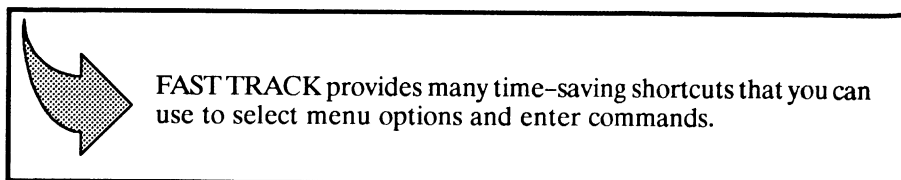
RH  
 PH  
 MainDH  
 MainDH  
 MainDD  
 MainDS  
 PF  
 RS  
 \*\*  
 \*\*  
 \*\*

In the Input Files window, the cursor is in the File Name field. Type **item**. Instead of typing the file name, you could use **CHOICES** (**ESC** **C**) to display a list of all the files in your database.

We'll skip the two remaining fields in the Input Files window for now. Press **GO** (**F1**) to accept the file name and return to the Report Writer edit screen.

Now that you have defined a file for the report, you can select fields from that file. Press **OPTIONS** (**CTRL-O**). When the horizontal menu of commands appears, select **INSERT→FIELD** (type **i** for Insert and then **f** for Field).

Instead of pressing **OPTIONS** (**CTRL-O**) and selecting **INSERT→FIELD**, you can use **ESC** **I**, which allows you to do the same thing — insert fields. FAST TRACK has many alternative key sequences, like this, that reduce the number of keystrokes it takes to select a menu option or enter a command. As you become familiar with FAST TRACK's menu structure and commands, you can use these shortcuts to define files, insert and delete fields, and return to the main menu — just to name a few.



FAST TRACK provides many time-saving shortcuts that you can use to select menu options and enter commands.

When you tell the Report Writer that you want to insert fields, it shows you a list of all the fields in the item file:

| Choices    |          |    |
|------------|----------|----|
| Files      | Fields   |    |
| ftdb.item  | Alloc    | RH |
| <variable> | Cost     | PH |
|            | Idesc    | H  |
|            | Item-num | D  |
|            | Iweight  | S  |
|            | Loc      | F  |
|            | Mnth-shp | S  |
|            | On-hand  | *  |
|            | Oorder   | *  |
|            |          | *  |
|            |          | ** |
|            |          | ** |
|            |          | ** |

Please choose a file.  
 ^O:Options F2:Help F4:Leave F9:Insert ^D:Delete  
 Report Name: itemrpt Section Title: Main-header Insert

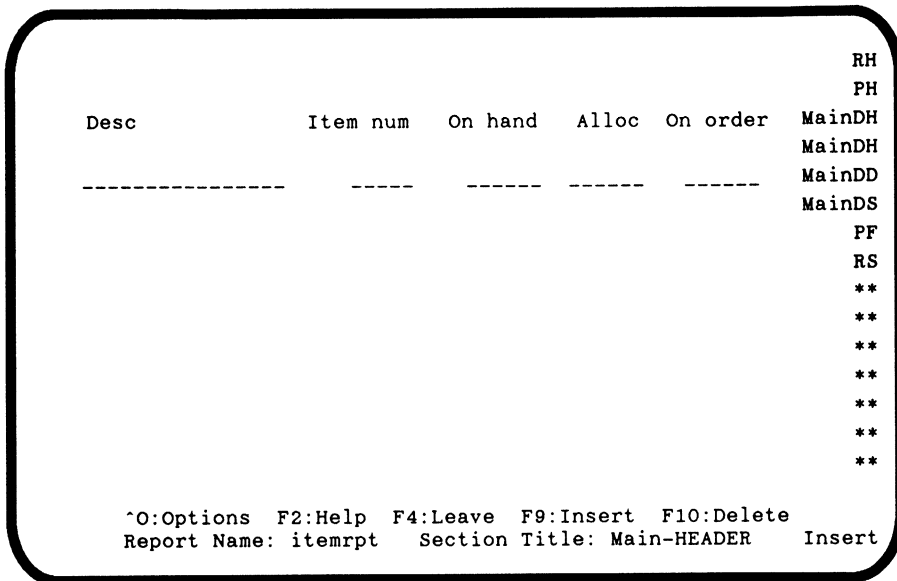
The Report Writer automatically lists your file as `ftdb.item` (the file is `item`, the database is `ftdb`). We want to insert the following fields into the report: **Idesc**, **Item-num**, **On-hand**, **Alloc**, **Oorder**. The Report Writer inserts fields in the order in which you choose them so you want to be sure to choose the above fields in the order in which there are listed.

To begin selecting the fields, press the right arrow key to move the highlight bar from the files column to the fields column. To select each of the five fields, use the down arrow or up arrow key to highlight the field and then press `RETURN`. Be sure to select the fields in the order mentioned above.



Each time you press **RETURN**, the Report Writer places an asterisk (\*) next to the selected field. If you make a mistake, you can deselect a field by highlighting it again and pressing **RETURN**.

When you have finished selecting the fields, press **GO** (F1). You'll see the five fields as column headings on the Report Writer edit screen.



Now that you have defined the file and fields for the item report, you can see how it looks on the screen. But before you do that, let's save the report. Press **OPTIONS** (**CTRL-O**) and select **COMMAND→SAVE**. This command saves your report as a file in the database. You'll see the message "Warning: If system crashes, work might be lost. Save permanently by going back to main-menu." at the bottom of your screen. This message indicates that you are in the middle of a transaction. Although **PROGRESS** writes your changes to the database when you save, these changes would be backed out if the system crashed and you would lose your changes. Since our data is not crucial, we can continue without going back to the main menu. Press **SPACEBAR** to continue.

Now, let's take a look at the item report. Press **OPTIONS** (**CTRL-O**) and select **COMMAND→VIEW**. This command generates a temporary report procedure that allows you to preview the report before finalizing it.

At the bottom of your screen, you will see messages informing you that the Report Writer is writing the report and then compiling it. When you see the message Press space bar to continue, the report is ready. Press  to display the item report.

| Desc            | Item num | On hand | Alloc | On order |
|-----------------|----------|---------|-------|----------|
| Fins            | 00001    | 0       | 80    | 2        |
| Tennis Racquet  | 00002    | 0       | 15    | 7        |
| Sweat Band      | 00003    | 142     | 79    | 65       |
| Cycle Helmet    | 00004    | 141     | 0     | 56       |
| Leotard, Black  | 00005    | 56      | 87    | 27       |
| Ski Poles       | 00006    | 15      | 8     | 5        |
| Buoyancy Vest   | 00007    | 37      | 18    | 15       |
| Runner's Vest   | 00008    | 38      | 0     | 12       |
| Swim Goggles    | 00009    | 0       | 22    | 3        |
| Bowling Bag     | 00010    | 37      | 34    | 9        |
| Lacrosse Stick  | 00011    | 51      | 0     | 15       |
| Rugby Shirt     | 00012    | 61      | 45    | 28       |
| Squash Glove    | 00013    | 0       | 15    | 16       |
| Knee Guards     | 00014    | 86      | 120   | 50       |
| Football Helmet | 00015    | 10      | 3     | 14       |
| Baseball Bat    | 00016    | 98      | 0     | 32       |
| Tennis Balls    | 00017    | 258     | 0     | 74       |
| Wet Suit        | 00018    | 10      | 14    | 4        |

Press space bar to continue.

Press  to scroll through the report. When you come to the end of the report, the Report Writer redisplay its edit screen. You can also cancel the display at any time by pressing  (F4).

Suppose that after viewing the report, you decide that you want to make some changes. That's easy enough.

### MODIFYING A REPORT

The item report displays data from all of the records in the Item file. Let's say you decide that the report should show only those records where the quantity on hand is less than 20. To do this, you set up *qualifications*, or conditions, that records must meet in order to be included in the report.

Qualifications are defined on the database file or files used in the report. Press  ( ) and select **DEFINE** → **FILES**. The Input Files window appears

with the name of the database, `ftdb`, and the name of the file, `item`, that you defined for the report.

| Input Files - section: Main |             |                 |
|-----------------------------|-------------|-----------------|
| Db Name                     | Parent Db   | Where           |
| File Name                   | Parent File |                 |
| <code>ftdb</code>           |             | <code>no</code> |
| <code>item</code>           |             |                 |

RH  
PH  
MainDH  
MainDH  
MainDD  
MainDS  
PF  
RS  
\*\*  
\*\*  
\*\*  
\*\*  
\*\*  
\*\*

ESC-C:Choices F1:Done F2:Help F4:Leave F9:Create F10>Delete  
Define Files  
Report Name: `itemrpt` Section Title: `Main-HEADER` Insert

Press `RETURN`. You'll see the Qualification window pop up on your screen.

Input Files - section: Main

|       | Db Name<br>File Name | Parent Db<br>Parent File | Where |        |
|-------|----------------------|--------------------------|-------|--------|
| Desc  | ftdb                 |                          |       | RH     |
| ----- | item                 |                          |       | PH     |
|       |                      |                          |       | MainDH |
|       |                      |                          |       | MainDH |
|       |                      |                          |       | MainDD |

| ( Field Name | Qualification                   | Compare | ) And/Or |
|--------------|---------------------------------|---------|----------|
|              | Field or Constant or Expression |         |          |

\*\*

In the PROGRESS language, the above is the same as this:

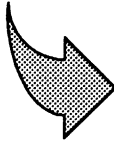
\*\*

FOR EACH item NO-LOCK.

ESC-C:Choices F1:Done F2:Help F4:Leave F9:Create F10>Delete F13:Free-form  
 Define Files  
 Report Name: Itemrpt      Section Title: Main-HEADER      Insert

Zero to two left parentheses for grouping expressions

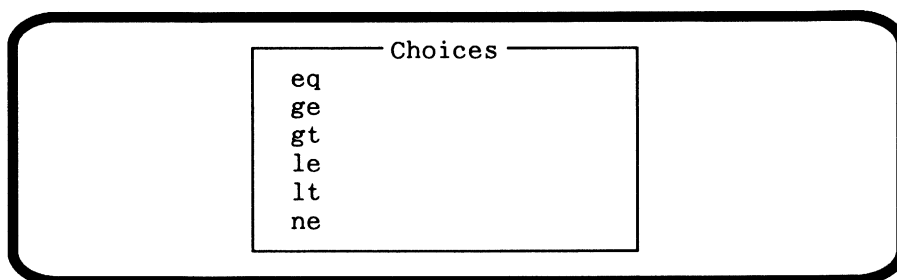
At any time, you can press **[HELP]** (F2) to access the FAST TRACK Help system. The Help system displays appropriate information about the task you are currently performing. To get Help information about the Qualifications window, press **[HELP]** (F2). You'll see a screen that explains how to enter qualifications. When you've finished reading the information, press **[END]** (F4) to return to the Qualifications window.



The FAST TRACK Help system provides context-sensitive help information whenever you need it.

Now, let's enter the qualification for the item file. Press **[RETURN]** to move the cursor to the Field Name column. Type **on-hand** and press **[RETURN]**. The cursor moves to the Compare field, and the Report Writer displays the comparison operator **eq** for equals. The **eq** operator is the default for fields that have numeric data.

In the item report, we want to display items that have less than 20 in stock. So let's change this comparison operator. Press **CHOICES** (**ESC** **C**) to see a list of the comparison operators you can use:



Use the down arrow key to highlight **lt** (less than) and then press **RETURN** to select it. You'll see **lt** in the Compare column of the Qualification window.

The cursor is in the column labeled Field or Constant or Expression. Type **20**. You have now entered a qualification that instructs the Report Writer to select only those records where the number on hand is less than 20.

Press **GO** (**F1**) to accept the qualification and return to the Input Files window. Notice that the Where field has a value of Yes. This indicates that the Item file now has a qualification defined for it. Press **GO** (**F1**) once again to accept the file definition and return to the Report Writer edit screen.

Now, save the changes you made to the item report. Press **OPTIONS** (**CTRL-O**) and select **COMMAND→SAVE**. Press **SPACEBAR** twice to go back to the menu.

Let's take a look at the modified item report. Press **OPTIONS** (**CTRL-O**) and select **COMMAND→VIEW**. When you see the message Press space bar to continue, press **SPACEBAR** to display the item report.

| Desc            | Item num | On hand | Alloc | On order |
|-----------------|----------|---------|-------|----------|
| Fins            | 00001    | 0       | 80    | 2        |
| Tennis Racquet  | 00002    | 0       | 15    | 7        |
| Ski Poles       | 00006    | 15      | 8     | 5        |
| Swim Goggles    | 00009    | 0       | 22    | 3        |
| Squash Glove    | 00013    | 0       | 15    | 16       |
| Football Helmet | 00015    | 10      | 3     | 14       |
| Wet Suit        | 00018    | 10      | 14    | 4        |
| Ski Wax, Red    | 00019    | 0       | 25    | 22       |
| Skis            | 00020    | 0       | 7     | 2        |
| Ski Bindings    | 00021    | 0       | 5     | 8        |
| Squash Racquet  | 00022    | 12      | 0     | 15       |
| Snorkel         | 00024    | 0       | 89    | 17       |
| Basketball      | 00034    | 18      | 12    | 5        |
| Bowling Shoes   | 00038    | 12      | 11    | 2        |
| Bowling ball    | 00039    | 9       | 12    | 4        |
| Ice Skates      | 00040    | 14      | 12    | 3        |
| Shuttle cocks   | 00044    | 4       | 0     | 2        |
| Golf shoes      | 00045    | 8       | 12    | 3        |

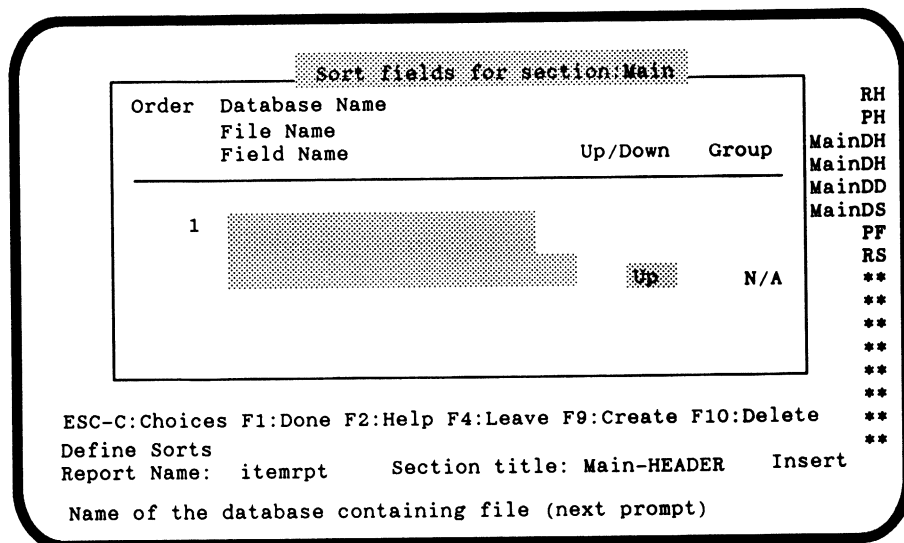
Press space bar to continue.

Your report now lists only those items that have less than 20 on hand.

### **SORTING DATA IN A REPORT**

The item report is sorted by item number. That's because, by default, the Report Writer uses the primary index to sort information in a report. It's easy to change the sort order for any FAST TRACK report, as you'll see in the next couple of paragraphs.

Press **OPTIONS** (**CTRL-O**) and select **DEFINE→SORTS**. Here's the screen that appears:



The Report Writer lets you sort data alphabetically or numerically in either ascending or descending order. You can sort on any field in your database, even if that field is not included in the report. It's also possible to define secondary sorts and to sort on a variable.

The default value, 1, appears in the Order column. This value indicates that the database field specified on this line is the primary sort field. To sort the item report alphabetically by the item descriptions, type **ftdb** for the Database Name, **item** for the File Name and **idesc** for the Field Name. Press **RETURN** after each one to go to the next line. Remember, you can also press **CHOICES** (**ESC** **C**) and select a value from the list of available choices.

The Up/Down column indicates the order of the sort. It can be either Up to indicate an ascending sort (A-Z) or Down to indicate a descending sort (Z-A). Since we want to sort the item report in A to Z order, leave the default value, Up, in the Up/Down column.

Press **GO** (**F1**) to enter the sort information and return to the Report Writer edit screen.

Now let's view the item report to see it sorted in alphabetical order by the item descriptions. Press **OPTIONS** (**CTRL-O**) and select **COMMAND→VIEW**. When you

see the message Press space bar to continue, press  to display the item report.

| Desc            | Item num | On hand | Alloc | On order |
|-----------------|----------|---------|-------|----------|
| Basketball      | 00034    | 18      | 12    | 5        |
| Bowling ball    | 00039    | 9       | 12    | 4        |
| Bowling Shoes   | 00038    | 12      | 11    | 2        |
| Fins            | 00001    | 0       | 80    | 2        |
| Football Helmet | 00015    | 10      | 3     | 14       |
| Golf clubs      | 00046    | 19      | 3     | 8        |
| Golf shoes      | 00045    | 8       | 12    | 3        |
| Ice Skates      | 00040    | 14      | 12    | 3        |
| Shuttle cocks   | 00044    | 4       | 0     | 2        |
| Ski Bindings    | 00021    | 0       | 5     | 8        |
| Ski Poles       | 00006    | 15      | 8     | 5        |
| Ski Wax, Red    | 00019    | 0       | 25    | 22       |
| Skis            | 00020    | 0       | 7     | 2        |
| Snorkel         | 00024    | 0       | 89    | 17       |
| Squash Glove    | 00013    | 0       | 15    | 16       |
| Squash Racquet  | 00022    | 12      | 0     | 15       |
| Stop watch      | 00055    | 12      | 0     | 5        |
| Swim Goggles    | 00009    | 0       | 22    | 3        |

Press space bar to continue.

Press  until you see the Report Writer edit screen again, or press  (F4) to cancel the display.

Let's say that the item report now looks just the way you want it to. What's next? You instruct FAST TRACK to generate a procedure that contains the PROGRESS statements needed to create the item report. Whenever a user selects option 3, Run Item Report, from the Item Menu, FAST TRACK runs the item report procedure.

To generate a procedure for your report, press  () and select **COMMAND→GENERATE**. First, you see a message asking you if you want the database prefix included in your report. Since we only have one database, we don't need the database name in our report. If you have more than one database, you should answer this question yes to avoid confusion. For now, type **no**. You now see messages telling you that FAST TRACK is writing and then compiling the report. When the procedure is complete, you'll see the message "The report is now generated (itemrpt.p)." To name the procedure, FAST TRACK adds the ".p" extension to your report name.



Press **SPACEBAR** to return to the Report Writer edit screen.

Now that your report is complete, you need to save it. Press **OPTIONS** (**CTRL-O**) and select **LEAVE→SAVE**. Your report is now saved in the database. **FAST TRACK** returns you to the Menu Editor where you'll see the Item Menu displayed.

## GENERATING A MENU PROCEDURE

Now that you're back in the Menu Editor, you can generate a **PROGRESS** procedure to run the Item Menu. Typically, a menu procedure is necessary only if you are creating a main menu. You may, however, want to generate a menu procedure for demonstration or testing purposes.

**FAST TRACK** allows you to generate a menu procedure even if you haven't assigned actions to all of the menu options, which is the case with the Item Menu.

Press **OPTIONS** (**CTRL-O**) and select **COMMAND→GENERATE**. The Menu Editor prompts you to enter a name for the menu procedure:

```
Enter a name for the output PROGRESS file
Command Generate > _____
Menu Name: Items
```

Type **itemmnu.p** and press **GO** (F1). The Menu Editor creates and saves a **PROGRESS** procedure, named **itemmnu.p**, that contains the code to run the Item Menu. You can use this procedure as you would any other **PROGRESS** procedure.

When you see the message "Menu procedure **itemmnu.p** generated," press **SPACEBAR**.

## LISTING AN OUTLINE OF YOUR MENU

As we mentioned at the beginning of this chapter, FAST TRACK lets you display information about your application so that you can track its development. To display an outline of your menu and the actions you've assigned, press **[OPTIONS]** (**[CTRL]-[O]**) and select **COMMAND→LIST**. Here's what you'll see on your screen:

```
Menu Titles and Labels [Name]

Item Menu
 List Items [Undefined choice type]
 Update Items [Undefined choice type]
 Run Item Report [Report itemrpt]
 Exit [Quit]

Press space bar to continue.
```

This outline shows you the title of your menu, each menu option, and the actions assigned to each option. When you're ready, press **[SPACEBAR]** to return to the Menu Editor.

Press **[OPTIONS]** (**[CTRL]-[O]**) and select Leave to return to the FAST TRACK main menu.

## YOU'RE NOW ON THE FAST TRACK!

From this brief introduction to PROGRESS FAST TRACK, you can see how this productivity tool can help you quickly prototype and modify the user interface for your applications. Using FAST TRACK menus, a few commands, and point-and-pick techniques, you have created and modified a menu, assigned actions to menu options, and generated a PROGRESS procedure to run your menu. You also designed a report and generated a procedure to run it.

Now that you've spent some time on the FAST TRACK to menus, forms, and reports, you probably want to know more about FAST TRACK's capabilities.

Included in your Test Drive package, you'll find the *FAST TRACK Tutorial* and the *FAST TRACK User's Guide*. The *FAST TRACK Tutorial* uses a step-by-step approach to introduce you to the FAST TRACK modules, including the Menu Editor, Screen Painter, Report Writer, and QBF Generator. The *FAST TRACK User's Guide* is a developer's guide and reference manual for the FAST TRACK interface. It includes detailed explanations of the features of each FAST TRACK module and provides information about maintenance of FAST TRACK objects and application deployment.

Go ahead and check out other FAST TRACK features. For example, you can use the Screen Painter to design a form for updating items and then use that form as the basis of a QBF (query-by-form) procedure that you create with the QBF Generator.

Remember, if you run out of sessions with the mydrive3 database, you can either start fresh with a new copy of the demo database (just as you did at the beginning of this chapter), or you can save the FAST TRACK objects you created in this chapter and use them in another ten sessions. If you want to save your FAST TRACK objects, refer to Appendix A.



CHAPTER 5

CAN THERE BE MORE FEATURES?





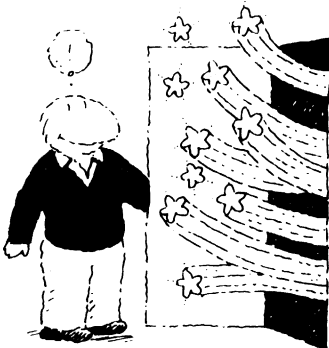
## CHAPTER 5

# CAN THERE BE MORE FEATURES?

Yes, it's true. There are more PROGRESS features. Along with the features you've already read about and tried out in the previous chapters, PROGRESS also has:

- A Procedure Library of “ready-to-use” PROGRESS procedures.
- ANSI-standard Structured Query Language (SQL).
- Multi-Database Programming.
- Gateways to ORACLE and RMS.
- Embedded SQL.
- The Host Language Call (HLC) interface for accessing C routines from within your PROGRESS procedures.

Read on to learn how these PROGRESS features can help meet your application development needs.



## THE PROGRESS PROCEDURE LIBRARY

The PROGRESS Procedure Library is an extensive set of PROGRESS procedures and include files that you can use to enhance your applications. These prewritten procedures can save you time and can streamline the process of building certain parts of your application.

The Library's menu-driven interface comes complete with the tools you need to locate and extract the procedures you want to use. The Library also provides you with menu options for updating, adding, and deleting procedures and for creating new libraries of your own.

Access to the Procedure Library is conveniently located on the PROGRESS Help menu. To display the Help menu, press **[HELP]** (F2) from the PROGRESS editor or select option 7 from the FAST TRACK main menu. Then type **p** and you'll see the Procedure Library's Main Menu.

The Procedure Library is dynamic — new procedures are added when they become available. Documentation for the Library, which is provided on-line so that it too can remain dynamic, includes an overview of the Library's contents, descriptions of the Main Menu options, and a tutorial. The on-line tutorial explains how to find a particular procedure in the Library, unpack it, and try it out. It also tells you how to extract a procedure from the library and place it in your application directory.

Many of the procedures in the Library include a "demonstration driver," or program, that allows you to actually run a procedure without unpacking it and setting up a test environment.

The Procedure Library is actually several libraries, each containing a set of procedures related to a particular topic. Within each library, you'll find additional documentation that explains the procedures stored in that library.



Here is a sampling of the types of procedures contained in the PROGRESS Procedure Library:

**Choose** Choose one or more elements from an array.  
Scroll through and optionally add, delete, or search for records.  
Scroll through text and search for a string.

**Pop-ups** Include a pop-up calculator, perpetual calendar, or a notepad in your application.

**Conversions** Convert data from one format to another.  
Transpose first and last names.  
Convert digits to words.

**Finance** Calculate the present or future value of an annuity.  
Include a loan amortization program.

**Math** Use expanded arithmetic and trigonometric functions.

**Science** Display a periodic table of elements and information about a selected element.

**Arrays** Create an array with up to ten dimensions.  
Calculate the sum of an array column or combination of columns.

**Tools** Produce a report that summarizes the structure of your database.  
Browse through SQL view definitions in the Dictionary.  
Compile procedures by wildcard and record output in a log file.  
Return the decimal and octal keycodes, the keyfunction, and the keylabel for a particular key.

**Operating System** Perform basic operating system file maintenance functions.

Append one operating system file to another.

Run the quoter utility to prepare data for input to a PROGRESS procedure.

## PROGRESS SQL

Another standard feature of PROGRESS is the support of ANSI-standard Structured Query Language (SQL). PROGRESS SQL includes the data definition language, queries, the data manipulation language, view definitions, and the granting and revoking of privileges.

The PROGRESS environment is designed to accept both SQL syntax and PROGRESS syntax. This means that you can use purely SQL statements, PROGRESS statements, or both within a single procedure. In addition, you can use either the SQL data definition language, or the PROGRESS Data Dictionary to define your database tables (files).

When you use PROGRESS SQL, you can take advantage of the high-productivity PROGRESS environment. For example, you can use the PROGRESS full-screen editor and all its editing functions to write your SQL statements. And then, with a single keystroke, the editor checks the syntax of your SQL statements and runs the procedure, just as it does when you enter PROGRESS 4GL statements. You also have access to the Data Dictionary where you can define or review your database structure.

In addition, you can use PROGRESS 4GL phrases within SQL statements to enhance the appearance of your query results on the screen and in printed reports. For example, you can determine frame attributes, such as the position of the frame on the screen and the number of records displayed in a frame at the same time. You can also use PROGRESS format phrases to specify such characteristics as column labels and where the data appears within the frame.

Within your SQL statements, you can use subscripts to access arrays that you have created with the PROGRESS 4GL. You can also use the PROGRESS date and logical data types in addition to the standard SQL data types.

## MULTI-DATABASE APPLICATIONS

Throughout this book, we used a single database, mydrive, mydrive2, or mydrive3. PROGRESS, however, has the capability to connect to as many as 240 databases from a single session. With this capability, you can develop reports that join data from different databases and procedures that update several database simultaneously. Not only can you connect to several PROGRESS databases at the same time, you can also access PROGRESS, ORACLE, and RMS databases located on different operating systems using several different networking protocols.

There are four ways to connect to a database:

- At PROGRESS startup with a PROGRESS startup command or in a startup file with the PROGRESS executable.
- From the PROGRESS editor or within an application with the CONNECT statement.
- With the PROGRESS Data Dictionary.
- Using the Auto-Connect feature.

In this section we are going to discuss connecting two PROGRESS databases with the CONNECT statement. The next section contains more information about connecting to ORACLE and RMS databases, and the *PROGRESS Programming Handbook* discusses the issues involved with multi-database programming.

To run the multi-database example, you need two databases. Let's use the databases you created in Chapters 3 and 4, mydrive2 and mydrive3. If you do not have copies of these two databases, create them with the following command (use the one appropriate for your operating system). The table uses mydrive2 as an example, use the name mydrive3 when you are ready to create that database.

| Operating System    | To Copy the Demo Database                                                              |
|---------------------|----------------------------------------------------------------------------------------|
| UNIX, DOS, and OS/2 | prodb mydrive2 demo                                                                    |
| VMS                 | PROGRESS/CREATE mydrive2 demo                                                          |
| BTOS/CTOS           | PROGRESS Create Database<br>New Database Name mydrive2<br>Copy From Database Name demo |

To start the PROGRESS session, refer to the following table and enter the command appropriate for your operating system.

| Operating System    | To Start PROGRESS                              |
|---------------------|------------------------------------------------|
| UNIX, DOS, and OS/2 | pro mydrive2                                   |
| VMS                 | PROGRESS mydrive2                              |
| BTOS/CTOS           | PROGRESS Single User<br>Database Name mydrive2 |

You are now in the PROGRESS editor. To connect to mydrive3, type the following command in the procedure editor and then press **GO** (F1):

```
CONNECT mydrive3 -1.
```

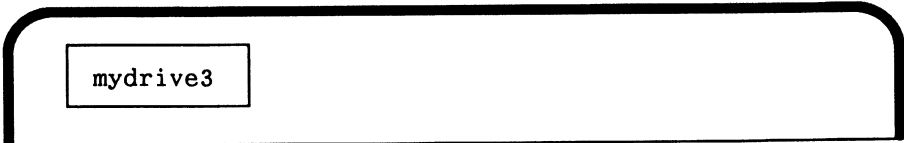
The -1 in the statement indicates that you are a single user.

When you have successfully connected, PROGRESS puts you back into the procedure editor (you may have to press **SPACEBAR** a couple of times if there are messages about the connection). To make sure the connection was successful, type the following command into the procedure editor and press **GO** (F1).

```
DISPLAY LDBNAME(2) .
```

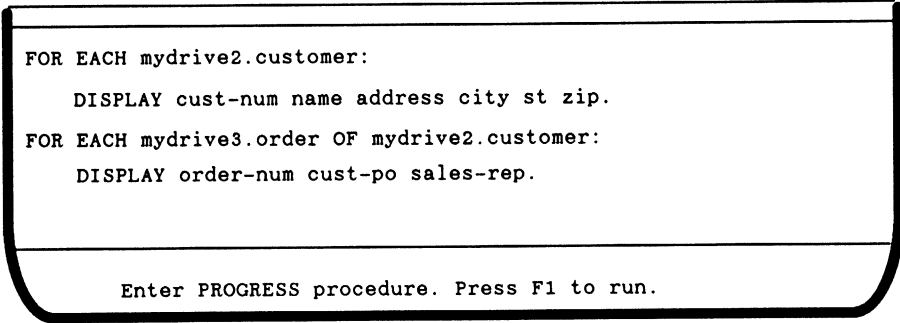
This statement tells PROGRESS to show you the logical database name (LDBNAME) of the second (2) database connected. The database you named at

startup (mydrive2) is considered the first database connected. If the connection was successful, you see the name of the database on your screen after you run this statement.



```
mydrive3
```

Now that you have access to two databases, you can begin running procedures that access information from both databases. For this procedure, we are going to assume that the mydrive2 database contains all the customer information and the mydrive3 database contains all the order information. We want to find out which customers from the mydrive2 database have orders on the mydrive3 database. To do this, enter the following procedure into the procedure editor.



```
FOR EACH mydrive2.customer:
 DISPLAY cust-num name address city st zip.
FOR EACH mydrive3.order OF mydrive2.customer:
 DISPLAY order-num cust-po sales-rep.
```

Enter PROGRESS procedure. Press F1 to run.

The upper case letters identify PROGRESS statements and the lower case letters are arguments you supply. This is simply for readability, your procedure can contain all upper case, all lower case, or a mixture of both. The indents are also for readability, they are not required. To save this procedure, press the F6 key and enter the procedure name at the prompt.

When you run this procedure, you will see the customer information stored in the mydrive2 database: the customer's identification number, name, address, and the contact at the customer site. On the same screen, you see the order information stored in the mydrive3 database: the order number of the customer from the mydrive2 database, that customer's purchase order number, and the sales representative of the customer.

When you press **GO** (F1), you see the following screen:

| Cust num | Name              | Addr          | City | State | Zip   |
|----------|-------------------|---------------|------|-------|-------|
| 1        | Second Skin Scuba | 79 Farrar Ave | Yuma | AZ    | 85369 |

| Ord num | Cust po | Sls rep |
|---------|---------|---------|
| 10      | PX A30  | SLS     |

Press **SPACEBAR** to page through the customers. Press **END** (F4) to get back to the editor. To go back to your operating system, type **quit** in the editor and press **GO** (F1).

## DATABASE GATEWAYS - ORACLE AND RMS

A database gateway lets you write PROGRESS applications that access existing PROGRESS databases, ORACLE databases, and RMS files. The PROGRESS application does not interfere with operation of previously developed non-PROGRESS applications. This saves time and money since you don't need to rewrite existing applications.

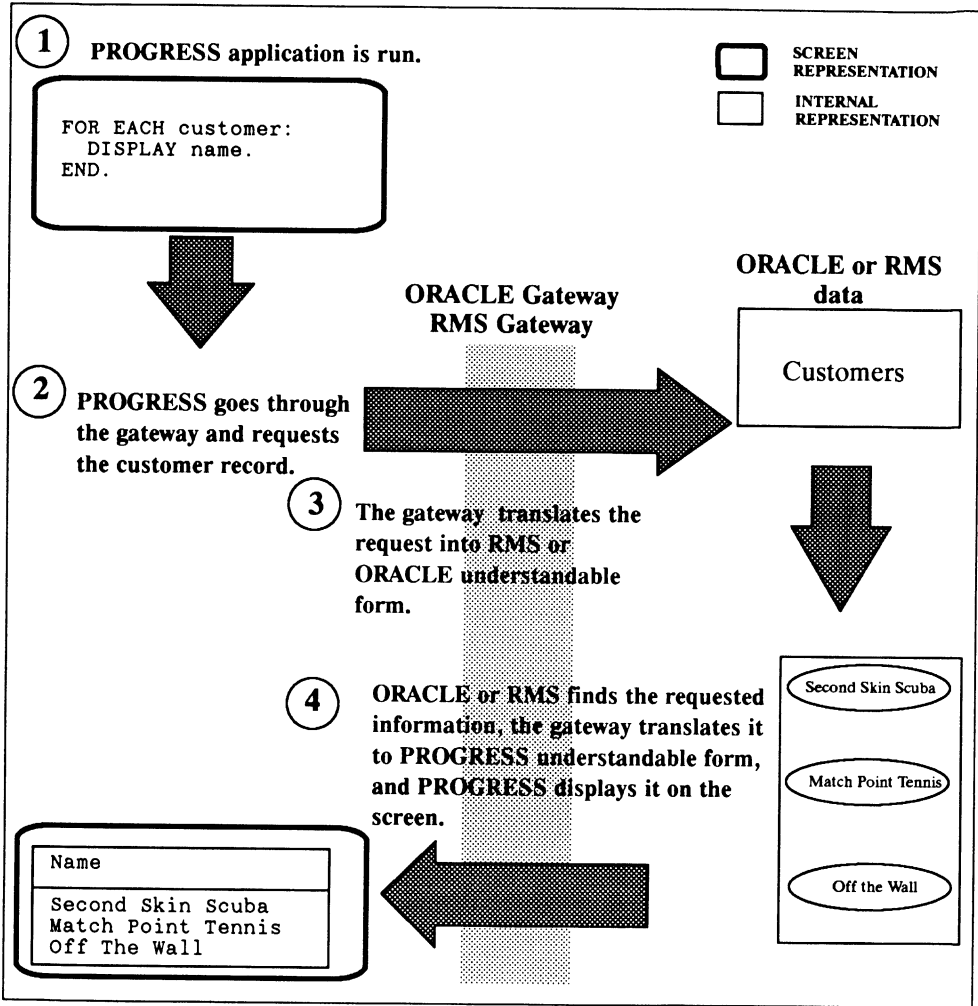
Using a database gateway involves copying the structure of the non-PROGRESS database or file system and placing the copy in a PROGRESS database called a *schema holder*. When you use the gateway to access the non-PROGRESS database, PROGRESS uses the information in the schema holder as a map to the data contained in the ORACLE database or the RMS files. Because the copy of the

ORACLE or RMS structure resides in a PROGRESS database, no operation performed within PROGRESS directly changes that structure.

With RMS, after you create the schema holder, you need to manually enter the field definitions or copy them from the Common Data Dictionary Plus (CDD +). With Oracle, the field definitions are included in the database structure so this step is not necessary.

After you copy the structure and define the fields (if necessary) you are ready to write applications.

When you execute a PROGRESS application that accesses an ORACLE database or an RMS file, the gateway translates the 4GL statements to their RMS or ORACLE equivalents. The application then finds the requested data. After the data is retrieved, the application returns to PROGRESS and continues. The following figure is a graphic representation of the gateway process.



## PROGRESS HOST LANGUAGE INTERFACE

The PROGRESS Host Language Interface (HLI) enables host language programs containing embedded SQL statements to access information in PROGRESS, ORACLE, and RMS databases. The implementation of embedded SQL with PROGRESS and the HLI is based on the definition set forth in the ANSI SQL86 standard.

The HLI includes a host language preprocessor that converts SQL statements into standard host language code that call database interface procedures. These



functions are called HLI entry points. HLI entry points include functions that open and close databases, log in to databases, connect to foreign databases, and retrieve error messages from the database log file.

## THE HOST LANGUAGE CALL INTERFACE

If you have invested time and effort in building C language routines, you don't have to worry about losing that investment. PROGRESS gives you a way to access those routines from the PROGRESS environment.

The Host Language Call (HLC) Interface provides the programmer with a method of calling C language functions from PROGRESS. This tool is particularly useful for accessing functions that the C language handles well, such as trigonometric functions or graphics routines.

HLC routines allow you to read data from PROGRESS, manipulate that data with the C language, and then write the manipulated data back to PROGRESS where it can be stored in a PROGRESS database. You can access data in PROGRESS shared buffers and variables and write data to shared buffers and variables.

Along with the HLC software, there is a demonstration program and a directory containing sample HLC applications that you can run to learn how HLC works. The *3GL Interface Guide* explains the fundamentals of HLC programming and describes the syntax for all of the HLC functions. This book is included when you purchase the PROGRESS 4GL/RDBMS.



CHAPTER 6

TECHNICALLY SPEAKING





# CHAPTER 6

## TECHNICALLY SPEAKING

Now that you have seen a PROGRESS application at work and have done some application development of your own, you can understand why building applications with PROGRESS is the most effective and efficient way to get your job done.

Now, for those of you interested in the particulars, here are some technical details about PROGRESS.

### HAVING WHAT IT TAKES

#### UNIX

If you are using UNIX, the machine resources you need to run PROGRESS depend on the particular UNIX machine you have. In general, you should have at least two megabytes of memory for the first user and an additional one megabyte of memory for each additional two to three users. You will need more memory if your version of the UNIX operating system is large. Additional memory will improve performance, especially if you will be running multi-user PROGRESS. PROGRESS can be used with most terminals.

#### DOS

If you are using a DOS system, your computer must have a hard disk, a floppy drive that can read 5.25 inch high density diskettes (1.2 MB) or 3.5 double density diskettes (720 KB). You must have 640K of memory.

#### OS/2

If you are using OS/2, your computer must have a hard disk, or a floppy drive that can read 3.5 inch diskettes (1.44 MB). You must have 640K of memory.

#### VMS

PROGRESS is designed to run on the VAX/VMS family of computers running VAX/VMS Version 5.0 or higher. It supports both DEC and non-DEC terminals.

## **BTOS/CTOS**

PROGRESS is designed to run in the BTOS/CTOS environment. In general, you should have a minimum of one megabyte of memory. If you want to access operating system utilities from within PROGRESS, you must also install the Context Manager software.

## **NETWORK SUPPORT**

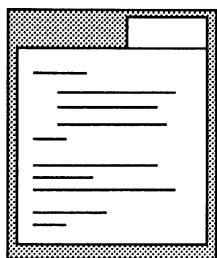
PROGRESS supports distributed processing over LANs, DECnet, and heterogeneous network environments.

LAN PROGRESS runs on DOS Local Area Networks. To run LAN PROGRESS, your configuration should have a machine to run the network file server, a machine to run the PROGRESS database server, and communication cards and LAN cables for each of the workstations on the network.

PROGRESS runs on computers connected by a DECnet-VAX network. Each machine on the network must be running the VMS operating system.

PROGRESS processing can also be distributed over many of the supported operating systems in a heterogeneous network. For example, applications running under DOS and XENIX can access a central database on a larger UNIX machine. This type of configuration offloads the front-end processing onto the smaller machines, allowing the larger machine to act as the database server.

## **PROGRESS COMPONENT SPECIFICS**



### **THE PROGRESS FOURTH-GENERATION LANGUAGE**

You've seen the incredible things you can do with the PROGRESS fourth-generation language. Here is a quick summary, by functional area, of many of the PROGRESS statements. For more details, see the *PROGRESS Language Reference* manual.

**RECORD PROCESSING**

|          |                             |
|----------|-----------------------------|
| CREATE   | Creates a new record.       |
| DELETE   | Deletes a record.           |
| FIND     | Finds a record.             |
| FOR EACH | Processes a set of records. |

**USER INTERFACE**

|            |                                                                                                     |
|------------|-----------------------------------------------------------------------------------------------------|
| DISPLAY    | Displays one or more fields from a single record or multiple records, or displays an entire record. |
| PROMPT-FOR | Prompts the user for input.                                                                         |
| SET        | Sets fields or an entire record.                                                                    |
| UPDATE     | Displays fields or an entire record, then lets the user update those fields or that record.         |
| CHOOSE     | Moves a highlight bar among a series of choices and allows the user to make a selection.            |

**PROGRESS** can automatically format screens and reports without separate formatting specifications. You do, however, have direct and comprehensive control over screen and report formatting.

When you use any of these user interface statements, you can choose many options and phrases to describe the way you want the screen or report to look. Here are just a few: **SIDE-LABELS**, **n COLUMNS**, **NO-LABELS**, **FRAME**, **FORMAT**, **LABEL**, **AT**, **TO**, **COLON**, **SPACE**, and **SKIP**.

### OTHER INPUT/OUTPUT

|                |                                                                      |
|----------------|----------------------------------------------------------------------|
| INPUT FROM     | Gets input from an ASCII file or UNIX or VMS device.                 |
| INPUT THROUGH  | Gets input from a concurrently executing UNIX process.               |
| OUTPUT TO      | Sends output to an ASCII file, the printer, or a UNIX or VMS device. |
| OUTPUT THROUGH | Sends output to a concurrently executing UNIX process.               |
| CALL           | Allows direct execution of programs written in C.                    |
| SEEK           | Positions index cursor to specified reload.                          |

### FLOW OF CONTROL

|                                  |                                                                                                        |
|----------------------------------|--------------------------------------------------------------------------------------------------------|
| IF...THEN...ELSE                 | Processes statements based on one or more conditions being true.                                       |
| REPEAT                           | Repeatedly iterates through a block or group of statements.                                            |
| LEAVE                            | Leaves a block.                                                                                        |
| NEXT                             | Does the next iteration of an iterating block.                                                         |
| RUN                              | Runs a separate PROGRESS procedure. Procedures can be run recursively.                                 |
| UNIX,DOS,VMS,<br>OS/2, BTOS,CTOS | Escapes into the operating system environment and returns to PROGRESS when you leave that environment. |

### DATABASE CONNECTIONS

|                            |                                                  |
|----------------------------|--------------------------------------------------|
| CONNECT                    | Connects a PROGRESS or non-PROGRESS database.    |
| DISCONNECT                 | Disconnects a PROGRESS or non-PROGRESS database. |
| CREATE } ALIAS<br>DELETE } | Reassigns logical database names.                |

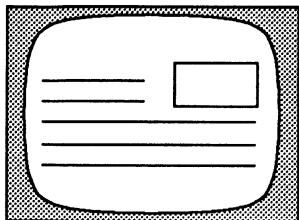


**ERROR RECOVERY**

- RETRY**                    Reprocesses a block of statements.
- UNDO**                    Undoes a group of statements, returning the database to the condition it was in at the start of that group of statements.

**FUNCTIONS (by category)**

- Aggregate Value**      ACCUM,    COUNT-OF,    MAXIMUM,    MINIMUM,    NUM-ENTRIES
- Arithmetic**            EXP, LOG, MODULO, RANDOM, ROUND, SORT, TRUNCATE
- Character**            BEGINS, CAPS, ENTRY, FILL, INDEX, KEYWORD, LC, LENGTH, LOOKUP, MATCHES, SUBSTRING
- Date**                    DATE, DAY, MONTH, TODAY, WEEKDAY, YEAR
- Data Conversion**      ASC, CHR, DECIMAL, INTEGER, STRING, TRIM
- Decision**              IF-THEN-ELSE, NOT, OR
- Record Status**        AMBIGUOUS,    AVAILABLE,    CAN-FIND,    FIRST,    FIRST-OF,    LAST,    LAST-OF,    LOCKED,    NEW,    RECID,    SEEK
- Screen Status**        ENTERED, FRAME-COL, FRAME-DOWN, FRAME-DB, FRAME-FIELD,    FRAME-FILE,    FRAME-INDEX,    FRAME-LINE,    FRAME-NAME,    FRAME-ROW,    FRAME-VALUE,    GO-PENDING,    INPUT,    MESSAGE-LINES,    NOT    ENTERED,    OVERLAY,    SCREEN-LINES
- System Status**        CAN-DO, CONNECTED, DBNAME, DBRESTRICTIONS, DBTYPE, GATEWAYS, KBLABEL, KEYCODE, KEYFUNCTION, KEYLABEL, LASTKEY, LDBNAME, LINE-COUNTER,    NUMALIASES,    OPSYS,    PAGE-NUMBER,    PAGE-SIZE,    PDBNAME,    PROGRAM-NAME,    PROGRESS,    PROPATH,    RETRY,    SDBNAME,    SEARCH,    SETUSERID,    TERMINAL,    TIME,    USERID



## PROGRESS FAST TRACK

You've experienced some of the ways you can use PROGRESS FAST TRACK to paint custom screens, menus, and reports. Here's a brief summary of the FAST TRACK modules. See the *PROGRESS FAST TRACK Tutorial* and the *PROGRESS FAST TRACK User's Guide* for more details.

### MENU EDITOR

- Create both top-level menus and submenus and define their characteristics.
- Design the overall structure of your application by assigning actions to menu options.
- Display a menu tree that shows the structure of your application.
- Generate a PROGRESS procedure to start your application.

### SCREEN PAINTER

- Create forms to display database information for the user, accept input from the user, or both.
- Use any number of database fields, variables, and text in your forms.
- Generate a PROGRESS FORM statement so that you can use your form in any PROGRESS procedure.

### REPORT WRITER

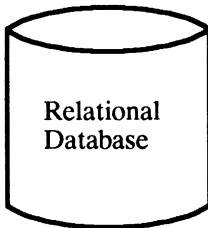
- Create reports using any number of database fields, variables, aggregates, and text.
- Set qualifications to determine the records that appear in your report.
- Sort the report data alphabetically or numerically in ascending or descending order.

- Generate a report procedure that you can use just as you would any other PROGRESS procedure.

### QBF GENERATOR

- Create query-by-form (QBF) procedures to perform file maintenance operations on your application database.
- Perform file maintenance operations on the records in one or more database files.
- Set criteria to determine the records that are queried.
- Use the QBF Generator's default form or create a custom form with the Screen Painter.

FAST TRACK also provides you with the tools needed to maintain the objects you create with FAST TRACK and the tools to deploy your application.



## THE PROGRESS RELATIONAL DATABASE MANAGEMENT SYSTEM

The PROGRESS Relational Database Management System consists of a central server (or broker on shared memory systems) for multi-user concurrency control. You can take advantage of PROGRESS' automatic concurrency features, or you can use the PROGRESS 4GL to override these features when necessary.

PROGRESS always gives you automatic recovery from a system failure. To provide the additional protection against media loss, you can also use PROGRESS' Roll Forward Recovery feature.

PROGRESS uses compressed B-tree indexing of multiple combined fields, in ascending or descending order, and can match on partial values.

PROGRESS lets you spread your database across multiple volumes so that the database size is not constrained by the size of a physical disk volume.

PROGRESS lets you restructure the database at any time by adding or deleting files, fields, and indexes.

PROGRESS lets you rename fields by using the global rename utility.

PROGRESS gives you high performance for transaction processing. Its database is fully relational for flexibility, data independence, and ease of use.

The use of the compressed B-tree indexing along with variable-length data storage methods, and a multi-threaded database architecture ensure efficient use of disk space and high-speed retrieval.

## LIMITS

### **Database**

Multiple gigabytes.

### **Files**

1023 per database (constrained by the number of indexes).

### **Indexes**

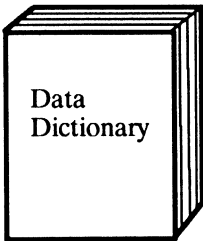
1023 per database. No explicit limit per file.

### **Records**

Approximately 32,000 bytes per record.

### **Fields**

Number per file constrained by record size.



## THE PROGRESS DATA DICTIONARY

The PROGRESS Data Dictionary lets you interactively define files, fields, and indexes.

The Data Dictionary recognizes a full range of data types:

**Character**

Constrained by the record and buffer size; full ASCII character set.

**Decimal**

50 digits maximum with up to 10 decimal places; optional European format for commas and decimal points.

**Integer**

-2,147,483,648 to +2,147,483,647

**Date**

1/1/32768 BC to 12/31/32767 AD (Including European format)

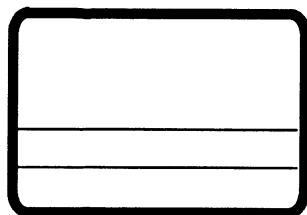
**Logical**

True/False; can optionally substitute any string value.

The Dictionary lets you define validation expressions and help messages for files. These definitions are applied, to maintain referential integrity, when a user tries to delete a record to which other records refer.

The Dictionary lets you define validation expressions and help phrases for fields. These definitions are applied automatically within data entry procedures.

The Dictionary provides a set of utilities to perform database management functions, such as dumping and loading database files, performing security administration activities, and accessing data exchange facilities.



## THE PROGRESS EDITOR

Writing PROGRESS procedures is easier and faster when you are using the PROGRESS full-screen editor.

### WHAT CAN YOU DO IN THE EDITOR?

Insert lines

Delete lines

Split lines

Join lines

Cut and paste blocks of text

Search for a string

Replace a string, individually or globally

Retrieve procedures

Save procedures

Recall the last procedure you ran

Clear the edit area

Move the cursor up, down, left, and right

Go to a specific line

Scroll up and down by page

Insert characters

Delete characters

In addition to all of these editing features, remember that the PROGRESS editor has a built-in syntax checking capability, and when the editor displays error messages, you can use Help to get context-sensitive information about those messages.

## WHAT ELSE IS THERE?

Now you know the kinds of things you can do with PROGRESS. You've run a PROGRESS application, written one of your own, used PROGRESS FAST TRACK, and had a close look at the components of PROGRESS.

What about other PROGRESS products? And what do you get when you buy PROGRESS? Read on.





CHAPTER 7

THE PROGRESS FAMILY OF PRODUCTS  
AND SERVICES





## CHAPTER 7

# THE PROGRESS FAMILY OF PRODUCTS AND SERVICES

You have seen how PROGRESS and PROGRESS FAST TRACK can help you develop and maintain high-performance applications.

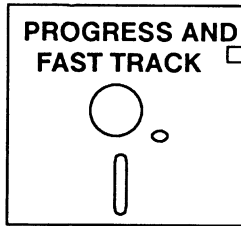
Other products in the PROGRESS family help you, as the application developer or as the end user, get the most out of your PROGRESS application. We also give you an extensive documentation set, comprehensive support services, training classes, and consulting services to meet your technical and educational needs.

### THE PROGRESS FAMILY OF PRODUCTS

So far, this book has introduced you to the PROGRESS Application Development System, which consists of the PROGRESS 4GL with SQL, the PROGRESS RDBMS, and PROGRESS FAST TRACK. Now let's take a look at the entire family of PROGRESS products.

- The PROGRESS Application Development System
- PROGRESS Query/Report
- PROGRESS Run-Time
- The Developer's Toolkit
- The PROGRESS/Oracle Gateway
- The PROGRESS/RMS Gateway
- PROGRESS HLI (Host Language Interface)

Once you develop a PROGRESS application, most likely you will want to distribute it to your end users. Depending on their requirements, your end users may not need to have the full PROGRESS Application Development System installed on their machines to run your application. End users can run an application if they have the PROGRESS Application Development System, PROGRESS Query/Report, or PROGRESS Run-Time. Each of these products offers end users varying degrees of functionality.



## THE PROGRESS APPLICATION DEVELOPMENT SYSTEM

The PROGRESS Application Development System combines the PROGRESS 4GL/RDBMS and the PROGRESS FAST TRACK Application Builder, each of which may be purchased separately. The PROGRESS 4GL/RDBMS provides you with the PROGRESS fourth-generation language, the PROGRESS relational database, the Data Dictionary, and the editor. PROGRESS FAST TRACK's menu-driven interface allows you to quickly design and develop integral parts of your application — menus, forms, reports, and QBF (query-by-form) procedures.

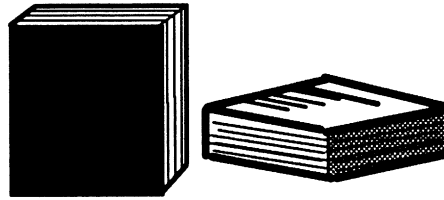
Because FAST TRACK is fully integrated with the PROGRESS environment, you can incorporate PROGRESS procedures into the applications you build with FAST TRACK. Conversely, you can incorporate FAST TRACK menus, forms, reports, and procedures into applications that you have written in the PROGRESS language.

The PROGRESS Application Development System also provides ANSI-standard SQL, allowing you to mix SQL and PROGRESS 4GL statements in the same procedure. In addition, you receive the PROGRESS Host Language Call (HLC) Interface for calling C routines from within your PROGRESS applications, and the PROGRESS Procedure Library with its "ready-to-use" procedures and files that can enhance any application.

The PROGRESS Application Development System provides developers with all the tools needed to build and run complex single- and multi-user applications across distributed databases..

As the end user of a PROGRESS application, this product provides you with all the capabilities the developer had when building the application. Having all these tools means that not only can you run applications, but you can also modify the application database and even write applications of your own.

## THE PROGRESS Documentation Set

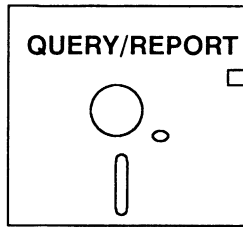


Progress Software Corporation publishes a comprehensive documentation set that not only teaches you how to use PROGRESS, but also thoroughly explains related areas, such as system administration and the environments in which PROGRESS runs. Both third-party reviewers and satisfied customers cite PROGRESS documentation as some of the best in the industry.

### The PROGRESS Application Development System comes with:

- Installation Notes
- PROGRESS Test Drive Manual
- PROGRESS Language Tutorial
- PROGRESS Programming Handbook
- PROGRESS Language Reference Manual
- Pocket PROGRESS
- PROGRESS FAST TRACK Tutorial
- PROGRESS FAST TRACK User's Guide
- PROGRESS System Administration Guide
- PROGRESS Environments Guide
- The 3GL Interface Guide

If you purchase one of the PROGRESS Gateway products, you will also receive the Database Gateway Guide.



## **PROGRESS QUERY/REPORT**

PROGRESS Query/Report is made up of the FAST TRACK Report Writer and most of the capabilities of the PROGRESS 4GL. The FAST TRACK Report Writer provides extensive ad hoc report writing capabilities through a menu-driven interface. With the Report Writer, you can quickly create different types of reports using the fields in one or more files of the application database.

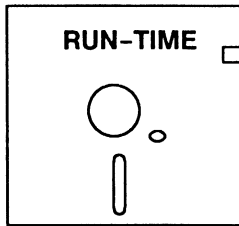
PROGRESS Query/Report is the full PROGRESS environment (PROGRESS 4GL/RDBMS) lacking only the capability of compiling procedures that change the database definitions or data. Query/Report includes the database, the syntax-checking editor, and limited versions of the PROGRESS 4GL and Data Dictionary that do not allow changes to the database.

PROGRESS Query/Report allows you, as the application developer, to deliver applications that permit end users to create reports and to write procedures but prevent them from writing procedures that modify the database.

As the end user of a PROGRESS application, having all the tools of PROGRESS Query/Report means that you can not only run the application, but you can create reports and ad hoc queries to the database. If you are familiar with SQL, you can also use PROGRESS SQL statements to query the database from the directly PROGRESS editor.

PROGRESS Query/Report comes with:

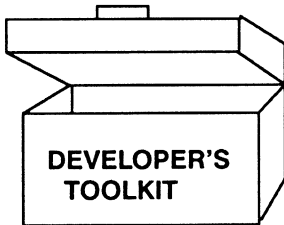
- Installation Notes
- PROGRESS FAST TRACK Tutorial
- PROGRESS FAST TRACK User's Guide



### PROGRESS RUN-TIME

PROGRESS Run-Time provides the environment needed to run PROGRESS applications, including the PROGRESS RDBMS. Because Run-Time does not include any of the tools used to write the application, Run-Time users can run a PROGRESS application but cannot write any of their own application procedures. PROGRESS Run-Time is ideal for distributing a turn-key PROGRESS application, and eliminates the risk of end users corrupting your source code.

PROGRESS Run-Time comes with Installation Notes.



### THE DEVELOPER'S TOOLKIT

When you develop applications that you plan to distribute to end users, you typically make several decisions. Do you want end users to be able to modify your application procedures and possibly add some of their own? Do you want end users to be able to access your database definitions and perhaps modify them? Do you want to distribute your application on a variety of machines and operating systems?

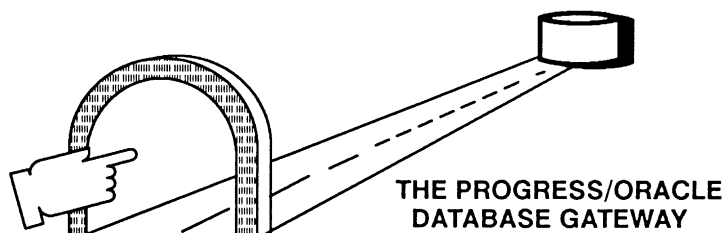
Then there are the additional tools you have to supply along with your application: a facility to dump and reload the database (if the user has PROGRESS Run-Time), batch files or scripts to start PROGRESS and access your applications, and so on.

The Developer's Toolkit consists of a complete set of tools for packaging and distributing an application or demo. The tools include:

- An empty database, so that you can modify the PROGRESS metaschema.
- A Toolkit Compiler, so that you can move your application to a different type of machine.
- A utility to encrypt your source procedures.
- Templates of the scripts, batch files, and command procedures that you must give to your users to start their application or dump and reload their database.

The Developer's Toolkit comes with the following documentation:

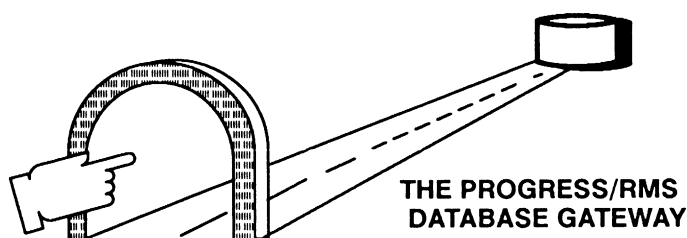
- Installation Notes
- The Developer's Toolkit Guide



Now you can use the PROGRESS 4GL on top of an Oracle database. By using the PROGRESS/Oracle Database Gateway, you can write PROGRESS applications that can run on top of the Oracle RDBMS. The PROGRESS/Oracle Database Gateway is a software module that translates the PROGRESS 4GL into SQL statements that can read and write the Oracle database.

Layering PROGRESS on an Oracle database makes most sense where there is already a large investment in Oracle database products. This is likely to be the case in some MIS departments, but VARs and consultants working with Oracle users will find the PROGRESS/Oracle combination extremely valuable as well.

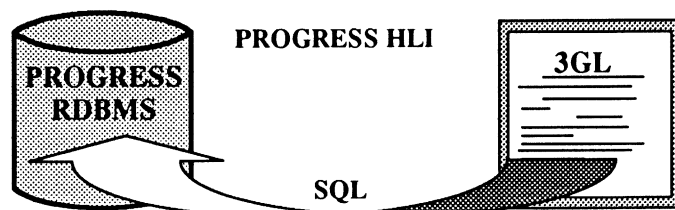




The PROGRESS/RMS Database Gateway is a software product that enables you to access Digital Equipment Corporation's RMS (Record Management Services) file information through PROGRESS applications.

Layering PROGRESS on RMS file structures makes most sense where there is already a large investment in DEC's RMS products. By using the PROGRESS/RMS Gateway, you can create applications using the PROGRESS 4GL, SQL, or the PROGRESS Host Language Interface, and have these applications read and write data from RMS. Although RMS is not a true relational database, the PROGRESS/RMS Gateway allows you to transfer RMS files into a PROGRESS RDBMS. This gives you all of the added value of the PROGRESS relational database, such as automatic record locking and crash recovery, while still protecting your investment in RMS.

7  
PROGRESS  
Products



Although you will want to develop new applications with the PROGRESS 4GL/RDBMS, you may already have an existing library of 3GL applications that need to interface with the PROGRESS relational database.

By using PROGRESS HLI (Host Language Interface), you can embed SQL statements into host language applications and retrieve or update records from PROGRESS databases. You can also use PROGRESS HLI to access Oracle and RMS databases as well. PROGRESS HLI includes a preprocessor that converts SQL statements into standard host language code that calls database interface procedures.

## WHEN YOU NEED SUPPORT

Progress Software Corporation offers its customer comprehensive support to help you get the most out of the PROGRESS Application Development System.



You can purchase several different kinds of support services including maintenance on your software products, classroom training, and consulting services.

### ANNUAL MAINTENANCE

Enrollment in the *Annual Maintenance Plan* allows you to keep up to date with new PROGRESS features and capabilities that are regularly introduced through software updates. With your enrollment in the plan, you also become eligible to receive telephone support. In addition, the plan provides you with regularly published editions of the *PROGRESS Technical Bulletin*, a booklet that describes issues related to your PROGRESS products.

### EDUCATION SERVICES

Progress Software offers courses designed to give you in-depth, hands-on training that helps you get the most out of PROGRESS. Courses are regularly held Progress Software's headquarters in Bedford, Massachusetts. Other training facilities are located across the United States and at Progress Software offices throughout the world. All training locations provide a computer-based training system, as well as computers for students to use during labs. You can also arrange to have PROGRESS training held at your location as well.

Each of the PROGRESS courses offers the opportunity to learn from experienced PROGRESS instructors and to solve actual application problems during programming labs. You'll also have a chance to consult with the instructors about issues that are specific to your application.

*Programming In PROGRESS* teaches the fundamentals of PROGRESS for beginning PROGRESS programmers. Students examine an actual application from database design, to setting data validation rules, to building and running a multi-user application.

The *Advanced PROGRESS Seminar* is designed for experienced PROGRESS programmers who have already developed sophisticated applications of their own. This course focuses on advanced programming techniques, PROGRESS internals, performance tuning, index strategies, and application design techniques.

The *PROGRESS FAST TRACK End User Seminar* is a two-day class that teaches end users how to expand the functionality of existing applications using the FAST TRACK Report Writer and Query-by-Form Generator.

The *PROGRESS FAST TRACK Developer's Seminar* explains how to quickly prototype applications using FAST TRACK. This course also examines how to use PROGRESS and FAST TRACK together to fine-tune and expand FAST TRACK applications beyond the basics.

The *PROGRESS System Administration Seminar* is designed for persons responsible for the system administration of PROGRESS and PROGRESS databases. This course focuses on common system administration requirements: backing up and restoring databases, using Roll Forward Recovery to restore databases after media loss or damage to a disk, running PROGRESS over a network, and configuring PROGRESS for a multi-volume environment. The *System Administration* course is offered in versions tailored to the various operating systems on which PROGRESS runs.

### **PROGRESS On-Line!**

PROGRESS On-Line! is an electronic bulletin board service that allows you to contact Technical Support as well as share information with Progress Software staff and customers 24 hours a day. To be eligible to use PROGRESS On-Line!, a customer must be enrolled in the Annual Maintenance Plan.

Connected to a searchable database, PROGRESS On-Line! acts as an encyclopedia. All the information in the Technical Bulletin is included in the PROGRESS On-Line! database. The bulletin board also contains user group dialogues, news of upcoming events, product announcements, and an up-to-date PROGRESS Application Catalog

### **CONSULTING SERVICES**

Progress Software Corporation also provides a broad range of consulting services. You can arrange to have a PROGRESS expert review your application from top to

bottom, focusing on areas such as schema design, coding standards, and program performance and tuning. This review can take place when your application is in the final testing stages or at intermediate steps in the development cycle, allowing you to easily incorporate suggested modifications. Fees for this type of consulting service are based on a per diem charge plus reimbursement of direct expenses.

For long term development projects, it is possible to have a PROGRESS expert in residence to act as an on-site “help desk.” This type of service can be provided for a minimum two-month commitment at a negotiated fee based on the length of the contract.

For more information about PROGRESS consulting services, contact the Manager of Consulting Services.

## **A NOTE ABOUT US**

Back in the infancy of fourth-generation language technology, a team of database and computer language specialists, recent graduates of the Massachusetts Institute of Technology, began pioneering new ways to develop multi-user applications. Their goal was to build a set of high-level database and application development tools that intermeshed smoothly.

That goal was attained with the creation of PROGRESS, the 4GL application development product that seamlessly integrates the tools needed to develop complete applications. VARs and MIS managers soon adopted PROGRESS with enthusiasm.

Today, there are over 40,000 PROGRESS licenses throughout the world. Progress Software Corporation’s installed base of applications ranges from those developed by individuals for their own use, to complex single-site applications, to multi-site government, manufacturing, financial, and retail applications. Users include Dannon Yogurt, Marriott Hotels, NCR Corporation, Sherwin-Williams Paint Stores, CitiBank, Pearle Vision, Hewlett-Packard, Pepsi-Cola, Consolidated Freightways, the Swedish Defense Ministry, the Australian Post Office, the Philadelphia Mint, and the U.S. Coast Guard.

Since its founding, Progress Software Corporation has averaged a 60% annual growth rate. A few minutes spent talking with PROGRESS customers will readily explain this impressive growth rate. For the third year in a row, PROGRESS has enjoyed greater user satisfaction than all of the other leading 4GLs on the market, according to surveys conducted by DATAPRO Research Corporation.

Progress Software's wholly-owned subsidiaries have offices in over a dozen countries in Europe and on the Pacific Rim. In addition, there are distributors and VARs throughout much of the rest of the world.

We know that, in producing the PROGRESS Application Development System, we have provided you with a complete environment that you can use to meet your application development needs. But we aren't stopping there. Our continuing goal is to bring you significant product enhancements, sophisticated development tools, and product services that help you and your users become even more productive.

## WHAT TO DO NEXT

You've reached the finish line! To start building applications with a full version of PROGRESS, call the PROGRESS office nearest you. In the U.S., you can call toll-free by dialing 1-800 FAST 4GL. See how your business can benefit by using the premier application development system.



APPENDIX A

SAVING DATABASE CHANGES





# APPENDIX A

## SAVING DATABASE CHANGES

With the Test Drive, you can access a demo database only ten times. This means that before you exit from the demo during the tenth session, you must make a decision — do you want to save any changes that you’ve made to the database?

### IF YOU DO NOT WANT TO SAVE YOUR CHANGES...

If you don’t want to save your changes, you can exit from PROGRESS. Then, if you want to create your own application or use PROGRESS FAST TRACK again, just start from the beginning of the appropriate chapter (Chapter 2 or 4) where you will make another copy of either the empty database or the demo database. When you make another copy of the empty database or the demo database and give it the same name as your first copy, PROGRESS informs you that a database by that name already exists and asks you if you want to replace it. Type *y* and press `RETURN` to replace your first copy of the database with the new copy.

### IF YOU DO WANT TO SAVE YOUR CHANGES...

If you *do* want to save your changes, you’ll have to perform some additional steps to dump your data before you exit from your tenth session. Then, when you restart PROGRESS with a new copy of the database, you’ll have to perform additional steps to load the data into the new database.

To save the changes you made to your copy of the empty database while writing your own application in Chapter 2, follow the steps in the sections titled “To Dump Dictionary Data,” “To Make a Copy of the Empty Database and Start PROGRESS,” and “To Load Dictionary Data.”

To save the changes you made while running FAST TRACK in Chapter 4, you must dump and then reload your FAST TRACK data as well as the Dictionary data. To do so, follow the steps in all of the following sections.

### TO DUMP FAST TRACK DATA

1. From the FAST TRACK main menu, select option **5**, Maintenance.
2. From the Fast Track Maintenance menu, choose option **2**, Dump FT Data Files.
3. FAST TRACK displays this prompt:

Dump FAST TRACK data files? no

Type **y** and press **[RETURN]**. FAST TRACK then displays a series of messages as it dumps the data files. When FAST TRACK has finished dumping the data files, you'll see the message **Press space bar to continue. Press [SPACEBAR]** and you'll return to the Fast Track Maintenance menu.

4. Press **[END]** (F4) twice to access the PROGRESS editor. Continue with the following section where you will use the Data Dictionary to dump data definitions and data.

### TO DUMP DICTIONARY DATA

From the Data Dictionary, you dump your data definitions and then your data. Begin by accessing the Data Dictionary.

1. In the PROGRESS editor, press **[CLEAR]** (F8) to clear the editor, if necessary. Then type **dict** and press **[GO]** (F1) to access the Data Dictionary.
2. From the Data Dictionary Main Menu, select Admin.
3. From the Admin menu, select **Dump Data and Definitions**.
4. From the Dump Data submenu, select **Data Definitions (.df file)**.
5. PROGRESS prompts you to select the files to dump. Type **all** and press **[RETURN]**.
6. PROGRESS then prompts you for the name of the file to store the data definitions. The default file name is the name of your database with the extension **.df** (for example, **mydrive3.df**). Press **[RETURN]** to accept the default file name.

As PROGRESS dumps the data definitions, it displays the name of each database file. When finished, PROGRESS returns you to the Data Dictionary Main Menu. Press **[RETURN]** to see the Admin menu.

7. From the Admin menu, select option Dump Data and Definitions.
8. From the Dump Data submenu, select File Contents.
9. Again, PROGRESS prompts you to select the files to dump. Press **[RETURN]** as many times as necessary to select all files. Press **[GO]** (F1) after all the files are selected.

PROGRESS then asks you to indicate in what directory you want the files dumped. Press **[RETURN]** to write to your working directory.

As PROGRESS dumps the files, it displays the name of each file. When all the files have been dumped, PROGRESS displays a table that shows what file was dumped, to what operating system file it was written, and the number of records dumped. Press **[SPACEBAR]** to go back to the Data Dictionary Main Menu.

10. Press **[END]** (F4) twice to return to the PROGRESS editor. Type **quit** and press **[GO]** (F1) to exit from PROGRESS.

**TO MAKE A COPY OF THE EMPTY DATABASE AND START PROGRESS:**

1. Whether you saved FAST TRACK data or only the Dictionary data, you now make a copy of the PROGRESS empty database. To do so, type the command for your operating system, as shown in the following table. For *dbname*, type the name you want to give your new database.

| Operating System | To Copy the Empty Database                                                                   |
|------------------|----------------------------------------------------------------------------------------------|
| UNIX             | prodb <i>dbname</i> empty                                                                    |
| DOS, OS/2        | prodb <i>dbname</i> empty                                                                    |
| VMS              | PROGRESS/CREATE <i>dbname</i> empty                                                          |
| BTOS/CTOS        | PROGRESS Create Database<br>New Database Name <i>dbname</i><br>Copy From Database Name empty |

If you saved FAST TRACK data, follow step 2. If you saved only the Dictionary data, follow step 3.

2. If you saved FAST TRACK data, start the Test Drive application using the copy of the empty database you just made. Enter the command for your operating system, as shown in this table:

| Operating System | To Start the Test Drive Application                                                                                                |
|------------------|------------------------------------------------------------------------------------------------------------------------------------|
| UNIX             | drive <i>dbname</i> -p prostart.p                                                                                                  |
| DOS, OS/2        | drive <i>dbname</i> -p prostart.p                                                                                                  |
| VMS              | @\$DISK:[DLCTDA]drive <i>dbname</i> /STARTUP = prostart.p                                                                          |
| BTOS/CTOS        | Submit<br>File List [Sys] < Sys > drive.sub<br>[Parameters] <i>dbname</i> -p prostart.p<br>[Force Expansion?]<br>[Show Expansion?] |

When you see the Test Drive banner, press  to access the PROGRESS editor. Skip step 3 and continue with the next section, "To Reload Dictionary Data."

3. If you saved only the Dictionary data, start PROGRESS using the new copy of the empty database. Enter the command for your operating system, as shown in this table:

| Operating System | To Start PROGRESS                          |
|------------------|--------------------------------------------|
| UNIX             | <i>pro dbname</i>                          |
| DOS, OS/2        | <i>pro dbname</i>                          |
| VMS              | PROGRESS <i>dbname</i>                     |
| BTOS/CTOS        | PROGRESS 4GL<br>[Options] -1 <i>dbname</i> |

When you see the PROGRESS banner, press SPACEBAR to access the PROGRESS editor.

**TO RELOAD DICTIONARY DATA:**

Before running any procedures or accessing FAST TRACK, you need to reload your Dictionary data.

1. To access the Data Dictionary from the PROGRESS editor, type **dict** and press GO (F1).
2. From the Data Dictionary Main Menu, select Admin.
3. From the Admin menu, select Load Data and Definitions.
4. From the Load Data submenu, select Data Definitions (.df file).
5. When PROGRESS prompts you for the name of the file that contains the data definitions, type the name of the file where you dumped the definitions (for example, mydrive3.df) and press RETURN twice.

When PROGRESS is finished loading the data definitions, it returns you to the Data Dictionary Main Menu. Press RETURN to access the Admin menu.

6. From the Admin menu, select Load Data and Definitions.
7. From the Load Definitions submenu, select File Contents (.d files). Press RETURN as many times as necessary to select all files. Press GO (F1) when all files are selected.

PROGRESS now asks you to indicate the directory you want to load the files from. Press **[RETURN]** if your files are located in your working directory, or enter the directory path where they are located. Press **[RETURN]** to accept the default 0 for the error rate. Press **[SPACEBAR]** when the load is complete.

8. Press **[END]** (F4) twice to return to the PROGRESS editor.

If you dumped FAST TRACK data, follow the steps in the next section to reload that data. If you did not dump FAST TRACK data, you can now write and run PROGRESS procedures from the editor.

#### TO RELOAD FAST TRACK DATA

1. In the PROGRESS editor, type **run ft.p** and press **[GO]** (F1) to start FAST TRACK.
2. From the FAST TRACK main menu, select option **5**, Maintenance.
3. From the Fast Track Maintenance menu, select option **3**, Load FT Data Files.
4. FAST TRACK displays this prompt:

Load FAST TRACK data files? no

Type **y** and press **[RETURN]**. FAST TRACK displays a series of messages as it loads the data files. When it has finished loading the files, you'll see the message **Press space bar to continue**. Press **[SPACEBAR]** to return to the Fast Track Maintenance menu.

5. Press **[END]** (F4) to return to the FAST TRACK main menu.

You can now modify the FAST TRACK objects you created in Chapter 4 or create new objects.

# GLOSSARY

**after-image file** — the file used to recover your database after a crash. The after-image file holds basically the same information as your database and also transaction notes needed to bring your database up to date after a crash.

**alias** — a synonym for a logical database name used within a PROGRESS procedure to reference a database. An alias is used as a database reference in PROGRESS procedures in place of a logical database name.

**application** — a computer program designed to perform a specific function, such as inventory control. A PROGRESS application has three parts: a place to store information, a way to get at the information, and the information itself.

**application security** — a means of protecting your database by allowing users to access only the data and procedures that they are authorized to access.

**argument** — the value that a function works on to produce its results.

**before-image file** — the file that ensures database integrity by keeping track of the status of your database in the event that a transaction is not completed or a system failure occurs.

**block** — a series of statements that PROGRESS treats as a single unit. Each block begins with a block header statement and concludes with an END. PROGRESS uses four kinds of blocks: FOR EACH, REPEAT, DO, and Procedure. A block can have one or more of the following properties: looping, record reading, clearing, transaction recovery, error processing, default screen design.

**character field** — an alphanumeric field that cannot be arithmetically manipulated. A character field may contain data of any kind. Although PROGRESS allows character fields of up to 255 characters, you will usually want to restrict the format length of a character field to the input/output line width of your terminal (typically 80 characters) by specifying the appropriate format.

**column** — a component of a record, and holds a value for the record. Also called a field.

**connection mode** — one of three possible ways a database is connected: single user, multi-user client, or multi-user direct access.

**data type** — a label assigned to a field that determines what type of data that field can store. PROGRESS supports five data types: character, integer, decimal, date, and logical.

**database gateway** — a PROGRESS add-on module that allows you to access a non-PROGRESS database such as ORACLE or RMS from within applications written in PROGRESS 4GL.

**data definition** — a term used to refer to the database structure, giving such information as name and where stored.

**Data Dictionary** — this term has two meanings. When capitalized, it refers to the menu-driven utility program. In lower case, data dictionary is a synonym for schema; that is, it refers to the actual database structure definitions. Typically in PROGRESS Data Dictionary refers to the PROGRESS program, while schema and data definitions are terms used to refer to the database structure.

**date field** — contains dates from 1/1/32768 BC through 12/31/32767 AD. You can specify dates in this century with either a two-digit year, such as 8/9/87, or a four-digit year (8/9/1987). Dates in other centuries require a four-digit year.

**decimal field** — a field that contains decimal numbers up to 50 digits in length. PROGRESS allows up to 10 digits to the right of the decimal point. If you enter blanks as the value of a decimal field, PROGRESS stores the value of that field as zero.

**directory** — a special system file containing a list of filenames and other associated information.

In BTOS/CTOS: a group of related documents, programs, and other associated information on a volume. A BTOS/CTOS directory only contains files; it cannot contain other directories. Directories on the BTOS/CTOS system do not expand as files are added. Therefore, it is important to create a directory that can accommodate all the files you want to put into it.

**editor** — a writing tool that allows you to create and edit PROGRESS procedures, checks the syntax of the procedure, and verifies the existence of files and fields named in the procedure.

**field** — a component of a record, a field holds a value for the record. Also called a column.

**file** — a collection of logically related records dealt with as a unit. For example, a payroll file is a collection of employee payroll records.

**function** — a process that generates a value. In PROGRESS functions are shortcuts to handling tasks as diverse as calculating the logarithm of an expression to determining the day of the week a particular date falls.



**global shared variable** — a variable that is available to all parts of a multi-part program and has the same value wherever it is used in the program.

**horizontal menu** — a menu style in which the menu options are in a line across the bottom of the screen.

**index** — a directory or table containing the fields identifying the records in a file and the locations where they are stored.

**integer field** — a field that contains whole numbers. They can be positive or negative, ranging in value from -2,147,483,648 through 2,147,483,647. If you enter blanks as the value of an integer field, PROGRESS stores the value of that field as zero.

**interactive SQL** — SQL statements in PROGRESS procedures. With interactive SQL, your procedures can contain SQL statements only, or both SQL and PROGRESS statement.

**logical database name** — a database reference representing the name of a connected physical database. The logical database name is used to resolve database references. That is, when a procedure is compiled against a database, it is the logical database name that is stored in the procedure's object code, and when a procedure executes, its database references must match the logical name of a connected database.

**logical field** — a storage area that contains yes/no or true/false values.

**null value** — an unknown or unavailable value for a particular column. The PROGRESS/SQL null value is equivalent to the PROGRESS unknown value (?).

**operator** — the symbol you use to perform calculations and comparisons such as numeric calculations, date calculations, character string manipulations, or data comparisons.

**overlay frame** — a frame that PROGRESS displays on top of other frames that appear at the same place on the screen.

**pathname** — the complete name of a directory or file starting at the root and tracing the hierarchy of the file.

**primary index** — usually the most frequently used indexed field. PROGRESS allows you to set one index as primary and then refers to it first when searching for a record.

**procedure** — instructions to the computer written primarily with the PROGRESS application language.

**record** — a collection of related items of data. Also called a row.

**relational database** — data stored in one or more files where one query can access information from one or more files in a single database.

**remote** — a terminal located at a distance from the computer communicating with the computer via telephone lines or other communication links.

**remote database** — a database located on a remote machine that is networked to the machine containing the application session.

**schema** — a description of a database's structure—the files it contains, the fields within the files, views, etc. In addition to database structure, PROGRESS database schemas contain items such as validation expression and validation message.

**schema-holder** — a PROGRESS database that contains the data definitions for one or more PROGRESS and non-PROGRESS databases.

**server** — a special mechanism that PROGRESS uses in a multi-user environment for coordinating the database access required by each user.

**string** — a character expression (a constant, field name, variable name, or any combination of these that results in a character string value).

**subschema** — a description of the structure of a non-PROGRESS database, held within a PROGRESS schema-holder. A subschema is created by the PROGRESS Data Dictionary using schema information found in the non-PROGRESS database's data dictionary files.

**table** — a collection of information organized into named columns. The following SQL statements define, modify, and delete tables: CREATE TABLE, ALTER TABLE, and DROP TABLE. An ORACLE *table* is equivalent to a PROGRESS *file*.

**unique index** — an indexed field where every value must be different. Social Security number could be a unique index.

**unknown value** — a special data type, represented by a question mark (?).

**validation expression** — a test to make sure that the user does not enter invalid data in a field.

**variable** — a temporary field for storing data.

**vertical menu** — a menu style in which the menu options are lined up one above the other.