1. INTRODUCTION

Purpose

This guide is designed to give you information about programming in a SYSTEM V/68 environment. It does not attempt to teach you how to write programs. Rather, it is intended to supplement texts on programming languages by concentrating on the other elements that are part of getting programs into operation.

Some of the features described in this guide may not be contained in the basic operating system software; they are provided as "unbundled" software. An example of this is Chapter 19; the Transport Interface is a part of the Network Supplement Extension software, which must be ordered separately.

Audience and Prerequisite Knowledge

As the title suggests, we are addressing programmers, especially those who have not worked extensively with this type of operating system. No special level of programming involvement is assumed. We hope the book will be useful to people who write only an occasional program as well as those who work on or manage large application development projects.

Programmers in the expert class, or those engaged in developing system software, may find this guide lacks the depth of information they need. For them we recommend the *Programmer's Reference Manual*.

Knowledge of terminal use, of an editor, and of the system directory/file structure is assumed. If you feel shaky about your mastery of these basic tools, you might want to look over the *User's Guide* before tackling this one.

Organization

The manual is divided into two parts that together include nineteen chapters, as follows:

Part 1:

- Chapter 1 — Overview

  Identifies the special features of the operating system that make up the programming environment: the concept of building blocks, pipes, special files, shell programming, etc. As a framework for the material that follows, three different levels of programming are defined: single-user, applications, and systems programming.

- Chapter 2 — Programming Basics

  Describes the most fundamental utilities needed to get programs running.

- Chapter 3 — Application Programming

  Enlarges on many of the topics covered in the previous chapter with particular emphasis on how things change as the project grows bigger. Describes tools for keeping programming projects organized.

Part 2:

- Chapters 4 through 19 — Support Tools, Descriptions, and Tutorials

  Include detailed information about the use of many of the operating system tools.

At the end of the text are appendices, a glossary, and an index.

The C Connection

SYSTEM V/68 supports many programming languages, and C language compilers are available on many different operating systems. Nevertheless, the relationship between the operating system and C has always been and remains close. Most of the code in SYSTEM V/68 is C, and over the years many organizations have come to use C for an increasing portion of their application code. Thus, while this guide is intended to be useful to you no matter what language you are using, you will find that, unless there is a specific language-dependent point to be made, the examples assume you are programming in C.

Hardware/Software Dependencies

The text reflects the way things work on a VME-based computer running SYSTEM V/68 at the Release 3 level. If you find commands that work a little differently in your system environment, it may be because you are running under a different release of the software. If some commands just don't seem to exist at all, they may be members of packages not installed on your system.

Notation Conventions

Whenever the text includes examples of output from the computer and/or commands entered by you, we follow the standard notation scheme that is common throughout the SYSTEM V/68 documentation:

- Commands that you type in from your terminal are shown in **bold** type.
- Text that is printed on your terminal by the computer is shown in `constant width` type. Constant width type is also used for code samples because it allows the most accurate representation of spacing. Spacing is often a matter of coding style, but is sometimes critical.
- Comments added to a display to show that part of the display has been omitted are shown in *italic* type and are indented to separate them from the text that represents computer output or input. Comments that explain the input or output are shown in the same type font as the rest of the display.

  Italics are also used to show substitutable values, such as *filename*, when the format of a command is shown.
- There is an implied RETURN at the end of each command and menu response you enter. Where you may be expected to enter only a RETURN (as in the case where you are accepting a menu default), the symbol <**CR**> is used.
- In cases where you are expected to enter a control character, it is shown as, for example, **CTRL-D**. This means that you press the **d** key on your keyboard while holding down the **CTRL** key.
- The dollar sign, **$**, and pound sign, #, symbols are the standard default prompt signs for an ordinary user and **root** respectively. A **$** means you are logged in as an ordinary user. A # means you are logged in as **root**.
- When the # prompt is used in an example, it means the command illustrated may be used only by **root**.

Command References

When commands are mentioned in a section of the text for the first time, a reference to the manual section where the command is formally described is included in parentheses — for example, **echo**(1). The numbered sections are located in the following manuals:

Section (1)	*User's Reference Manual*
Sections (1, 1M), (7), (8)	*System Administrator's Reference Manual*
Sections (1), (2), (3), (4), (5)	*Programmer's Reference Manual*

Information in the Examples

While every effort has been made to present displays of information just as they appear on your terminal, it is possible that your system may produce slightly different output. Some displays depend on a particular machine configuration that may differ from yours. Changes between releases of the operating system software may cause small differences in what appears on your terminal. Appendix A describes the command packages available on VME-based computers. If you find yourself trying to execute a nonexistent command, check Appendix A, then talk to your system administrator.

Where complete code samples are shown, we have tried to make sure they compile and work as represented. Where code fragments are shown, while we can't say that they have been compiled, we have attempted to maintain the same standards of coding accuracy for them.

The Scope of This Guide

The *Programmer's Guide* is about tools used in the process of creating programs in a SYSTEM V/68 environment.

Tools not covered in this text:

- the **login** procedure
- SYSTEM V/68 editors and how to use them
- how the file system is organized and how you move around in it
- shell programming

Information about these subjects can be found in the *User's Guide* and numerous other commercially available texts.

Tools covered here can be classified as follows:

- utilities for getting programs running
- utilities for organizing software development projects
- specialized languages
- debugging and analysis tools
- compiled language components that are not part of the language syntax — for example, standard libraries, and system and function calls.

The Shell as a Prototyping Tool

Any time you log on to the system, you are using the shell. The shell is the interactive command interpreter that stands between you and the operating system kernel. Because of its ability to start processes, direct the flow of control, field interrupts, and redirect input and output, it is a full-fledged programming language. Programs that use these capabilities are known as shell procedures or shell scripts.

Much innovative use of the shell involves stringing together commands to be run under the control of a shell script. The dozens and dozens of commands that can be used in this way are documented in the *User's Reference Manual*. Time spent with the *User's Reference Manual* can be rewarding. Look through it when you are trying to find a command with just the right option to handle a knotty programming problem. The more familiar you become with the commands described in the manual pages, the more you will be able to take full advantage of the operating system environment.

It is not our purpose here to instruct you in shell programming. What we want to stress here is the important part that shell procedures can play in developing prototypes of full-scale applications. While understanding all the nuances of shell programming can be a complex task, getting a shell procedure up and running is far less time-consuming than writing, compiling, and debugging compiled code.

This ability to get a program into production quickly is what makes the shell a valuable tool for program development. Shell programming allows you to "build on the work of others" to the greatest possible degree, since it allows you to piece together major components simply and efficiently. Many times even large applications can be done using shell procedures. Even if the application is initially developed as a prototype system for testing purposes rather than being put into production, many months of work can be saved.

With a prototype for testing, the range of possible user errors can be determined—something that is not always easy to plan out when an application is being designed. The method of dealing with strange user input can be worked out inexpensively, avoiding large recoding problems.

An available operating system tool can accomplish with a couple of lines of instructions what might take a page and a half of compiled code. Shell procedures can intermix compiled modules and regular operating system commands to let you take advantage of work that has gone before.

Three Programming Environments

We distinguish among three programming environments to emphasize that the information needs and the way in which operating system tools are used differ from one environment to another. We do not intend to imply a hierarchy of skill or experience. Highly-skilled programmers with years of experience can be found in the "single-user" category, and relative newcomers can be members of an application development or systems programming team.

Single-User Environment

Programmers in this environment are writing programs only to ease the performance of their primary job. The resulting programs might well be added to the stock of programs available to the community in which the programmer works. Single-user programmers may not have externally imposed requirements, or coauthors, or project management concerns. The programming task itself drives the coding directly. One advantage of a timesharing system is that people with programming skills can be set free to work on their own without having to go through formal project approval channels and perhaps wait for months for a programming department to solve their problems.

Single-user programmers need to know how to:

- select an appropriate language
- compile and run programs
- use system libraries
- analyze programs
- debug programs
- keep track of program versions

Most of the information required to do these functions at the single-user level can be found in Chapter 2.

Application Environment

Programmers working in this environment are developing systems for the benefit of other, non-programming users. Most large commercial computer applications still involve a team of applications development programmers. They may be employees of the end-user organization or they may work for a software development firm. Some of the people working in this environment may be more in the project management area than working programmers.

Information needs of people in this environment include all the topics in Chapter 2, plus additional information on:

- software control systems
- file and record locking
- communication between processes
- shared memory
- advanced debugging techniques

These topics are discussed in Chapter 3.

Systems Environment

Programmers in this environment are engaged in writing software tools that are part of, or closely related to, the operating system itself. The project may involve writing a new device driver, a data base management system, or an enhancement to the operating system kernel. In addition to knowing their way around the operating system source code and how to make changes and enhancements to it, they need to be thoroughly familiar with all the topics covered in Chapters 2 and 3.

1