INTERACTIVE
A Kodak Company

386/ix™

product family

386/ix Software Development System Guide

First printing (December 1988)

INTERACTIVE Systems Corporation
2401 Colorado Avenue, 3rd Floor
Santa Monica, California 90404

Contents

Contents

386/ix Software Development System Guide

CONTENTS

How To Use the 386/ix Software Development System Guide

INTRODUCTION

Welcome to the *386/ix Software Development System Guide*. The instructions in this guide provide the information you need to install the 386/ix™ Software Development System extension and to write applications or device drivers that will be integrated with the 386/ix Operating System. It also explains how to use the documentation that accompanies the 386/ix Software Development System optional extension.

What's Included

The *386/ix Software Development System Guide* includes:

- **Overview of the 386/ix Software Development System**
  Provides an introduction to the 386/ix Software Development System and its components.
- **386/ix Software Development System Installation Instructions**
  Provides step-by-step instructions on how to install the 386/ix Software Development System extension.
- **Integrating Software With the 386/ix Operating System**
  Provides information needed to write application packages and device drivers that are to be integrated with the 386/ix Operating System.
- **Reader's Comment Form**
  Provides you with a way to tell us what you like or dislike about this guide and to send us your ideas for making it even better.

Other documentation provided with the 386/ix Software Development System includes:

- **AT&T UNIX V.3.2 Programmer's Guide**
  Provides information needed to write and maintain programs in the UNIX® System V/386 operating system environment.
- **AT&T UNIX V.3.2 Programmer's Reference Manual**
  Provides manual entries of interest to programmers working in the UNIX System V/386 operating system environment.
- **AT&T UNIX V.3.2 SDS Release Notes**
  Provides information about the content and uses of the 386/ix Software Development System.
- **AT&T UNIX V.3.2 Integrated Software Development Guide**
  Provides information for independent software vendors who plan to develop UNIX system application programs and installable drivers to run on 386 computer systems. It supplements the information found in "Integrating Software With the 386/ix Operating System" in this guide and should be used in conjunction with that document.

Optional documents available to support the 386/ix Software Development System optional extension are:

- **AT&T UNIX V.3.2 Network Programmer's Guide**
  Provides information needed to write networking applications in the UNIX System V/386 operating system environment.
- **AT&T UNIX V.3.2 STREAMS Primer**
  Provides a technical overview of STREAMS for managers and developers who are experienced with UNIX systems and networking or other data communication facilities.
- **AT&T UNIX V.3.2 STREAMS Programmer's Guide**
  Provides information for programmers who plan to use STREAMS in a UNIX System V/386 operating system environment.

Where to Begin

If you are installing the 386/ix Software Development System . . .
First, read the overview to learn about the components of the 386/ix Software Development System. Then, read and follow the instructions in "386/ix Software Development System Installation Instructions."

If you are new to UNIX . . .
Read the first three chapters of the *Programmer's Guide*. Then, refer to the chapters that are appropriate for your software development project.

If you are an experienced UNIX programmer . . .
Depending upon your knowledge of UNIX, refer as necessary to the *Programmer's Guide* and *Programmer's Reference Manual* that accompanied your 386/ix Software Development System extension.

If you want to write applications or drivers that will be integrated with the 386/ix Operating System . . .
Read the *Integrated Software Development Guide* and "Integrating Software With the 386/ix Operating System."

If you are writing networking applications . . .
Refer to the *STREAMS Primer*, *STREAMS Programmer's Guide*, and *Network Programmer's Guide*.

Conventions Used

Throughout this guide, boxed words indicate keys on your keyboard. For example, RETURN refers to the key that moves the cursor to the next line. When you are instructed to type a command, the command must always be followed by using the RETURN key.

☛ Keys on your keyboard may be labeled differently than those shown in this guide. For example, the RETURN key is labeled ENTER on some systems. If your hardware or software vendor supplies additional documentation with your system, read that documentation for information on key names before you continue with this guide.

When a sequence of keystrokes using the CTRL key is listed, use the CTRL key as you would the SHIFT key. Hold down the CTRL key, and while it is down, press the next key (or keys) specified. For example, to use the sequence CTRL `s`, you would hold down the CTRL key while typing the `s` key.

Illustrations of computer screen displays, file names, directory names, and commands are printed in a typeface called `constant width`. Constant width text looks like the text produced by most typewriters. Whenever you are instructed to type anything shown in constant width in this guide, type it exactly as it is shown.

Italics indicate the variables in a command or instruction format. In actual use, a real name or number replaces the italicized text. For example, the sequence `rm` *filename* shows the format for removing a file. The word *filename* is replaced with the name of a real file that you would like to remove from your system. Italics are also used for emphasis and when new terminology is introduced.

Numbers preceded by the symbol § refer to section numbers within that document.

FOR MORE INFORMATION

The documentation included in this guide provides information about how to install the 386/ix Software Development System. It supplements information found in the *Programmer's Guide*, *Programmer's Reference Manual*, *Integrated Software Development Guide*, *STREAMS Primer*, *STREAMS Programmer's Guide*, and *Network Programmer's Guide*. For a complete listing of all 386/ix-related documentation, refer to the "Documentation Roadmap" included in the *386/ix Operating System Guide*.

Overview

Overview

Overview of the 386/ix Software Development System

INTRODUCTION

The 386/ix™ Software Development System extension is designed for programmers who plan to use the 386/ix Operating System as a software development environment. This extension consists of the C Software Development Set and the Extended Terminal Interface. The C Software Development Set contains the programs needed to compile, link, and debug C programs. It also contains the Source Code Control System (SCCS), `make`, `yacc`, `lex`, and other software development tools. The Extended Terminal Interface contains a set of libraries to assist the rapid development of screen management applications.

Who Can Use the 386/ix Software Development System?

Any programmer who wants to develop programs that run on the 386/ix or UNIX® System V.3.2 operating systems can use the 386/ix Software Development System extension. You do not need to have extensive experience with the UNIX operating system to develop programs to run under UNIX, but you should be familiar with the UNIX file and directory structure, UNIX commands, and a UNIX editor. Although many UNIX applications are written in the C programming language, you may develop programs in other languages, such as COBOL, FORTRAN, and Pascal, if a compatible compiler is installed on your system.

The 386/ix Software Development System is particularly useful to those programmers who:

- Want to develop or enhance C language programs.
- Need tools for advanced programming and symbolic debugging.

386/ix is a trademark of INTERACTIVE Systems Corporation.
UNIX is a registered trademark of AT&T.

- Want to track and maintain serial versions of files and programs.
- Want to work with shared libraries.
- Want to optimize interactive, character-oriented C applications.

C SOFTWARE DEVELOPMENT SET

The C Software Development Set contains the tools and utilities listed in the following sections. They assist in the development of efficient C programs, advanced programming, symbolic debugging, and keeping a history of source code files.

C Programming Language Development Tools

- **C Compiler**
  The C compiler supports and extends the C language. Its extensions include arbitrary length names for variables and function names, structure assignments and arguments, functions returning structure values, enumerated data types, multiple external variable declarations, assembly language escapes from C, insertion of arbitrary strings into object modules, floating point support, data type *void*, and additional preprocessor directives.

- **`cc` Command and `cpp` Preprocessor**
  The `cc` command controls the phases of compilation and automatically calls the C preprocessor, assembler, and link editor phases. It accepts files containing C source code as input and yields an executable module named `a.out`. It accepts source files with assembly language code and passes them directly to the assembler. The C preprocessor (`cpp`) performs file inclusion and macro substitution.

- **Optimizer**
  The optimizer is an optional component that makes compiler-generated assembly language code more efficient. It reduces space requirements and speeds the execution time of the resulting object code.

- **Assembler and Assembly Language**
  The assembler (`as`) converts assembly language code into a relocatable object module and provides access to predefined macros. It is useful for developing applications that require close interaction with hardware.

- **Link Editor**
  The link editor combines relocatable object modules and libraries to produce either an absolute, executable load module in Common Object File Format (COFF) or a relocatable object file for use in subsequent link edits. It performs relocation, resolves external references, and adds symbolic debugging information into its output file.

- **Object File Manipulation Tools**
  These tools consist of a number of utilities for reading and manipulating object files, including `ar`, `cprs`, `dis`, `dump`, `lorder`, `nm`, `size`, and `strip`.

- **Libraries**
  These are libraries for object files, access to system calls, input/output, string manipulation, mathematical functions, and memory allocation.

Advanced Programming Tools and Utilities

- **Programming and Debugging Utilities**
  The programming and debugging utilities assist in the design and development of applications and systems. Some of the programming utilities include `cxref`, `ctrace`, `lint`, `sdb`, `make`, `lex`, and `yacc`. Refer to the *Programmer's Guide* and the *Programmer's Reference Manual* for more information about these utilities.

- **Productivity Utilities**
  Three utilities are available to enhance C program efficiency: `cscope`, `lprof`, and `prof`. `cscope` is an interactive program that searches a program for functions, function calls, macros, and variables, and allows you to view and edit those portions of the code. The two profilers, `lprof` and `prof`, perform dynamic program analysis or an analysis at runtime. They provide information about the program code in an easy-to-view format. Refer to the *Programmer's Guide* for more information about these utilities.

- **Source Code Control Utilities**
  The Source Code Control System (SCCS) is a collection of utilities designed to assist in the management of large scale software and documentation development projects. SCCS organizes changes to files into a series of versions and revisions of the text, allowing users to access the precise version that they need. SCCS can restrict changes to the files to permit only users on a specific list to modify each file. It also prevents several users from trying to change the same version of a file at the same time. SCCS logs every file change for tracking purposes.

EXTENDED TERMINAL INTERFACE

The Extended Terminal Interface is a set of libraries that aid in the rapid development of screen management applications. The libraries enable the programmer to incorporate screen management and data entry capabilities into programs. It includes the following libraries:

- `Curses/Terminfo` **Low-Level Function Library**
  This includes routines for writing character-oriented screen management applications that are independent of the type of terminal.
- **High-Level Function Libraries**
  These libraries include functions that create, manipulate, and display panels, forms, and menus. They are built atop `curses`.
- **Terminal Access Method (TAM) Transition Library**
  The library enables character mode applications developed for the UNIX PC using the Terminal Access Method to run on other processor/terminal configurations.

FOR MORE INFORMATION

For additional information on each of the individual tools and data files in this extension, refer to the *SDS Release Notes*, *Programmer's Guide*, and *Programmer's Reference Manual* described in the "Documentation Roadmap" in the *386/ix Operating System Guide*.

386/ix Software Development System
Installation Instructions

CONTENTS

386/ix Software Development System Installation Instructions

1. INSTALLING THE 386/ix SOFTWARE DEVELOPMENT SYSTEM

The 386/ix™ Software Development System extension is installed on your fixed disk and requires about 3.75 MB of space. It requires that the Kernel Configuration and File Management subsets already be installed on your machine.

1. To begin the installation, use the System Administration command, `sysadm`, or log in as `sysadm` to access the Main Menu. Your screen will look similar to this:

```
                    SYSTEM ADMINISTRATION

  1 diskmgmt        disk management menu
  2 filemgmt        file management menu
  3 machinemgmt     machine management menu
  4 packagemgmt     package management menu
  5 softwaremgmt    software management menu
  6 syssetup        system setup menu
  7 ttymgmt         tty management menu
  8 usermgmt        user management menu

Enter a number, a name, the initial part of a name, or
? or <number>? for HELP,  q to QUIT:
```

2. Type 5 to access the Software Management menu. Your screen will then look similar to this:

386/ix is a trademark of INTERACTIVE Systems Corporation.
UNIX is a registered trademark of AT&T.
Intel is a registered trademark of Intel Corporation.
XENIX is a registered trademark of Microsoft Corporation.

```
                    SOFTWARE MANAGEMENT

  1 installpkg      install new software package onto built-in disk
  2 listpkg         list packages already installed
  3 removepkg       remove previously installed package from built-in disk
  4 runpkg          run software package without installing it

Enter a number, a name, the initial part of a name, or
? or <number>? for HELP,  ^ to GO BACK,  q to QUIT:
```

3. Select option 1, `installpkg`. The system prompts you to insert the first diskette into the diskette drive. The screen will look similar to this:

```
Insert the removable medium for the package you want to
install into the diskette drive.
Press <RETURN> when ready.  Type  q  to quit.
```

4. Insert the first 386/ix Software Development System diskette into the diskette drive. The system asks you to confirm that this is the package you want to install. Use **RETURN** to start the installation process.

```
Install the Software Development System-Version 1.3 package? (y):
Installing the Software Development System.
Copyright (c) 1987 AT&T
All Rights Reserved
The following files are being installed:
/bin/as
/bin/cc
    .
    .
    .
/lib/libPW.a
/usr/options/sd.name
1790 blocks
Floppy diskette number 1 is complete
Remove floppy and insert floppy number 2
Type <return> when ready:
```

5. Remove the first diskette and insert the second one. Then use **RETURN** to continue.

```
The following files are being installed:
/lib/libc.a
/lib/libc_s.a
      .
      .
      .
/usr/bin/delta
/usr/options/sd.name
1895 blocks
Floppy diskette number 2 is complete
Remove floppy and insert floppy number 3
Type <return> when ready:
```

6. Repeat this process until all the diskettes have been inserted and installed. The final screen will look similar to this:

```
The following files are being installed:
/usr/lib/libns.a
/usr/lib/libc.a
        .
        .
        .
/usr/lib/yaccpar
/usr/options/sd.name
1684 blocks
Floppy diskette number 4 is complete
Installation of the Software Development System-Version 1.3
is complete.
You may now remove the medium from the diskette drive.
```

(The names of some of the files have been omitted for the sake of brevity.)

The 386/ix Software Development System is now installed on your fixed disk.

2. THE 386/ix SOFTWARE DEVELOPMENT SYSTEM FILES

The following is a complete list of the files delivered with the 386/ix Software Development System. The program and command files are described; additional files are listed at the end.

`/bin/as`
The assembler.

`/bin/cc`
A program that provides the interface to the C Compilation System.

`/bin/conv`
The common object file program that converts byte order.

`/bin/convert`
A program that converts 5.0 archive files to 5.2 archive file format.

`/bin/cprs`
A program that reduces the size of a common object file by removing duplicate structure and union descriptors.

`/bin/dis`
The object code disassembler.

`/bin/dump`
A program that dumps selected parts of each of its object file components.

`/bin/gencc`
A program that allows the user to interactively create a new `cc` front-end.

`/bin/ld`
The link editor for common object files.

`/bin/list`
A program that produces a C source listing with line number information from a common object file.

`/bin/lorder`
A program that finds the ordering relationship for an object library.

`/bin/make`
A program used to maintain, update, and regenerate groups of programs.

`/bin/nm`
A program that prints the symbol table of each common object file.

`/bin/size`
A program that produces section size information in bytes for each loaded section in the common object files.

`/bin/strip`
The command file that removes the symbol and line number information from a common object file.

`/etc/install`
A program used to install a file in a specific place within a file system.

`/lib/comp`
The C compiler.

`/lib/cpp`
The C language preprocessor.

`/lib/crt0.o`
A start-up file provided for backwards compatibility.

`/lib/crt1.o`
Runtime start-up file.

`/lib/crtn.o`
Runtime start-up file.

`/lib/libPW.a`
The PW library.

`/lib/libc.a`
The standard library.

`/lib/libc_s.a`
The shared standard library.

`/lib/libld.a`
The common object file access routines library.

`/lib/libm.a`
The math library.

`/lib/libm1167.a`
The math library for Weitek 1167 chips.

`/lib/mcrt0.o`
Runtime start-up files with profiling code.

`/lib/mcrt1.o`
Runtime start-up files with profiling code.

`/lib/optim`
The optimizer.

`/usr/bin/admin`
A program used to create new SCCS files and to change the parameters of existing ones.

`/usr/bin/as386.sed`
A `sed` script that converts an Intel® ASM386 assembler source file to a source file acceptable to the UNIX® system `as` assembler.

`/usr/bin/cb`
A program that reads C programs and writes them to the standard output with spacing and indentation that display the structure of the code.

`/usr/bin/ccoff`
A command file that converts a COFF file by byte-swapping all multi-byte integers in the file.

`/usr/bin/cdc`
A program used to change the delta comment of a named SCCS file.

`/usr/bin/cflow`
A program that analyzes a collection of C, `yacc`, `lex`, assembler, and object files, and attempts to build a graph charting the external references.

`/usr/bin/comb`
A program used to generate a shell procedure that will reconstruct given SCCS files.

`/usr/bin/ctrace`
The C program debugger, which allows you to follow the execution of a C program, statement by statement.

`/usr/bin/cxref`
A program that analyzes a collection of C files and builds a cross-reference table.

`/usr/bin/delta`
A program used to make permanent changes in the named SCCS file.

`/usr/bin/get`
A program used to generate an ASCII text file from a named SCCS file.

`/usr/bin/help`
The `help` command file.

`/usr/bin/lex`
A program used to generate programs to be used in simple lexical analyses of text.

`/usr/bin/lint`
A program used to check C programs for features that are bugs, nonportable, or wasteful.

`/usr/bin/m4`
A macro processor intended as a front end for C and other languages.

`/usr/bin/mcs`
A program used to manipulate the comment section of an object file.

`/usr/bin/omf`
A program used to convert an object module from COFF format to XENIX® OMF format.

`/usr/bin/prof`
A program used to interpret and display the profile data produced by the `monitor` function.

`/usr/bin/prs`
A program used to print all or part(s) of an SCCS file to the standard output in a user-supplied format.

`/usr/bin/regcmp`
A program used to compile the regular expressions in a given file and place them in an output file.

`/usr/bin/rmdel`
A program used to remove the most recent delta to a named SCCS file.

`/usr/bin/sact`
A program used to inform the user of any impending changes to a given SCCS file.

`/usr/bin/sccsdiff`
A program used to compare two versions of a given SCCS file and generate the differences between them.

`/usr/bin/sdb`
The symbolic debugger used with C and F77 programs.

`/usr/bin/tsort`
A program that prints to the standard output a totally ordered list of items consistent with a partial ordering of items mentioned in the given input file.

`/usr/bin/unget`
A program that nullifies the effect of the `get` program used prior to creating an intended new delta to a given SCCS file.

`/usr/bin/val`
A program used to determine whether or not a given SCCS file meets the characteristics specified by the user.

`/usr/bin/vc`
A program used to copy lines from the standard input to the standard output. User-declared keywords may be replaced by their string *value* when they appear in plain text and/or control statements.

`/usr/bin/what`
A program used to identify the given SCCS file(s).

`/usr/bin/yacc`
A compiler-compiler that generates an `lr` parsing algorithm from a context-free grammar.

Include files:

`/usr/include/a.out.h`

`/usr/include/agent.h`

`/usr/include/alarm.h`

`/usr/include/aouthdr.h`

`/usr/include/ar.h`

```
/usr/include/assert.h
/usr/include/core.h
/usr/include/ctype.h
/usr/include/dial.h
/usr/include/dirent.h
/usr/include/dumprestor.h
/usr/include/errno.h
/usr/include/execargs.h
/usr/include/fatal.h
/usr/include/fcntl.h
/usr/include/filehdr.h
/usr/include/ftw.h
/usr/include/grp.h
/usr/include/ieeefp.h
/usr/include/ldfcn.h
/usr/include/limits.h
/usr/include/linenum.h
/usr/include/macros.h
/usr/include/malloc.h
/usr/include/math.h
/usr/include/math.h3b5x
/usr/include/memory.h
/usr/include/mnttab.h
/usr/include/mon.h
/usr/include/nan.h
/usr/include/nlist.h
/usr/include/nsaddr.h
/usr/include/nserve.h
```

```
/usr/include/pn.h
/usr/include/poll.h
/usr/include/prof.h
/usr/include/pwd.h
/usr/include/regexp.h
/usr/include/reloc.h
/usr/include/rje.h
/usr/include/scnhdr.h
/usr/include/search.h
/usr/include/setjmp.h
/usr/include/sgtty.h
/usr/include/signal.h
/usr/include/stand.h
/usr/include/stdio.h
/usr/include/storclass.h
/usr/include/string.h
/usr/include/stropts.h
/usr/include/strselect.h
/usr/include/symbol.h
/usr/include/syms.h
/usr/include/sys
/usr/include/termio.h
/usr/include/time.h
/usr/include/tiuser.h
/usr/include/tp_defs.h
/usr/include/ttysrv.h
/usr/include/unistd.h
/usr/include/ustat.h
```

```
/usr/include/utmp.h
/usr/include/values.h
/usr/include/values.h3b5x
/usr/include/varargs.h
```

Data files and tools:

```
/usr/bin/ctc
/usr/bin/ctcr
/usr/lib/ctrace/runtime.c
/usr/lib/dag
/usr/lib/flip
/usr/lib/help/ad
/usr/lib/help/bd
/usr/lib/help/cb
/usr/lib/help/cm
/usr/lib/help/cmds
/usr/lib/help/co
/usr/lib/help/de
/usr/lib/help/default
/usr/lib/help/ge
/usr/lib/help/he
/usr/lib/help/lib
/usr/lib/help/lib/help
/usr/lib/help/lib/help2
/usr/lib/help/prs
/usr/lib/help/rc
/usr/lib/help/un
/usr/lib/help/ut
/usr/lib/help/vc
```

```
/usr/lib/lex/ncform
/usr/lib/lex/nrform
/usr/lib/libcrypt.a
/usr/lib/libg.a
/usr/lib/libgen.a
/usr/lib/libl.a
/usr/lib/libmalloc.a
/usr/lib/libns.a
/usr/lib/libp/libc.a
/usr/lib/libp/libm.a
/usr/lib/libp/libm1167.a
/usr/lib/libp/libmalloc.a
/usr/lib/libsln.a
/usr/lib/liby.a
/usr/lib/lint1
/usr/lib/lint2
/usr/lib/llib-lc
/usr/lib/llib-lc.ln
/usr/lib/llib-lm
/usr/lib/llib-lm.ln
/usr/lib/llib-lmalloc.l
/usr/lib/llib-port
/usr/lib/llib-port.ln
/usr/lib/lpfx
/usr/lib/nmf
/usr/lib/xcpp
/usr/lib/xpass
/usr/lib/yaccpar
```

Integrating Software With the 386/ix Operating System

CONTENTS

Integrating Software With the 386/ix Operating System

1. INTRODUCTION

Using subsets as a standard distribution medium offers a consistent way of installing software products, whether they are composed of a few simple files or the programs that make up a set of complicated device drivers. This document provides guidelines for building 386/ix™ subsets with the Software Integration Tools that accompany your 386/ix Software Development System. It explains a variety of procedures, from the simple, such as storing a few files on a diskette, to the complex, such as making a device driver into an installable package. An overview of the various phases of device driver module development, culminating with the module's interface with the 386/ix kernel, is also given. This document is meant to provide 386/ix-specific information that complements the more general information in the *Integrated Software Development Guide*, which also accompanies the 386/ix Software Development System.

Although this information is intended for Independent Software Vendors (ISVs) and Independent Hardware Vendors (IHVs) who want to make sure that their application or device driver can easily be installed on the 386/ix Operating System, programmers may also find this information to be useful. Documentation updates for driver subsets will be provided as appropriate with future releases of the 386/ix Operating System. Please check your documentation before installing drivers on new releases.

1.1 Before You Begin

This document assumes that you have already installed the 386/ix Operating System on your computer and that you are familiar with

---

the general principles of computer operation. If you have never used a UNIX® operating system before, you should read the "UNIX Primer" that accompanied your 386/ix Operating System and Appendix A of this document.

1.2 Overview of This Document

This document is divided into eight major sections:

1. INTRODUCTION
   This section provides a general overview of this document.
2. INTEGRATING APPLICATIONS WITH THE 386/ix OPERATING SYSTEM
   This section describes the structure of a 386/ix subset, how they may be installed on the 386/ix Operating System, and how to use the Software Integration Tools to make your own subsets so that your customers or co-workers can easily install your software on the 386/ix Operating System. It also describes what the tools do.
3. INTEGRATING DEVICE DRIVERS WITH 386/ix SUBSETS
   This section discusses the special case of software that contains device drivers. It explains how to make 386/ix subsets that will automatically install the driver modules in the appropriate directories on the fixed disk, so they can be installed using the 386/ix Operating System `kconfig` utility. Additional information about installable device drivers can be found in the *Integrated Software Development Guide* that accompanies your 386/ix Software Development System extension.
4. UPDATING 386/ix DEVICE DRIVERS FROM RELEASE 1.0.6
   This section discusses the differences between 386/ix Operating System Release 1.0.6 and Release 2.0 as they apply to device driver modules and their location on the system. It describes the actions necessary to make a driver object that works with Release 1.0.6 work with Release 2.0 of the 386/ix Operating System.
5. APPENDIX A: USING DISKETTES WITH THE 386/ix OPERATING SYSTEM
   For your convenience, this section describes how to use diskettes with the 386/ix Operating System. It lists the special file names used for various types of diskettes and

illustrates the use of some standard UNIX utilities that may be useful when integrating your software.

6. APPENDIX B: SAMPLE SHELL SCRIPTS
   For your convenience, this section provides sample shell scripts to use in constructing subsets.

7. APPENDIX C: BUILDING A SUBSET WITHOUT THE TOOLS
   This section describes how to make your own 386/ix subsets without using the tools that accompany the 386/ix Software Development System.

8. APPENDIX D: RELATED MANUAL ENTRIES
   386/ix manual entries that are of interest when integrating software with the 386/ix Operating System are included for your convenience.

2. INTEGRATING APPLICATIONS WITH THE 386/ix OPERATING SYSTEM

INTERACTIVE delivers a special set of tools to help make it easy for you to integrate your applications with the 386/ix Operating System. Integrating your application using the tools will make it easy for your customers to install your application on 386/ix systems.

These tools are delivered with the 386/ix Software Development System on the diskette labelled *Software Integration Tools*. This section describes the utilities available with 386/ix for installing software, describes the content and form of a typical 386/ix subset, and explains how to use INTERACTIVE's special tool kit to make your applications easily installable. It also explains the tasks that are performed automatically by the tools.

2.1 Utilities for Installing Software on 386/ix

Applications and utilities available for the 386/ix Operating System are usually distributed on diskettes or tape. Several utilities are currently available on the 386/ix Operating System to load or unload software from the distribution media. Some are standard UNIX utilities, such as `cpio` and `tar`, provided with every UNIX implementation. You must know how to use UNIX commands and device naming conventions to use these utilities.

Other value-added utilities for loading and unloading software that have been developed to help make the 386/ix Operating System easier to understand and use are currently available on the 386/ix Operating System. The value-added utilities supplied with the 386/ix Operating System are:

- `sysadm`
  386/ix subsets are distributed on diskettes as UNIX file systems that can be mounted using the `sysadm` utility. `sysadm` is the name of a UNIX command as well as a set of utilities developed by AT&T for UNIX System V Release 3.0. It provides a consistent menu-driven interface for the UNIX system administrator. The procedures for adding users, hooking up a terminal, or installing an application are all performed similarly. The 386/ix Operating System contains an adapted and enhanced version of the `sysadm` utility that makes it convenient to add most enhancements to standard UNIX. Although AT&T no longer supports the `sysadm` utility for UNIX System V/386 Release

3.2, on which 386/ix Release 2.0 is based, INTERACTIVE continues to support it. It is strongly recommended that you use INTERACTIVE's tools to integrate your software with the 386/ix Operating System. Applications integrated with these tools are much easier for customers to install.

- `installpkg`
  Instead of the `sysadm` utility, AT&T's UNIX System V/386 Release 3.2 now uses a utility called `installpkg`. If you have software packaged for such a system, you will also be able to install it using `sysadm`.

- `custom`
  Release 2.0 is the first release of the 386/ix Operating System based on the version of UNIX that offers XENIX® binary compatibility. Most XENIX applications that are commercially available are supplied on diskettes that need to be installed using a utility called `custom`. This program is supplied with the 386/ix Operating System for compatibility with XENIX applications. The `installpkg` option of the 386/ix `sysadm` utility has been enhanced to recognize diskettes that use `custom`, and it automatically calls `custom` when needed.

To insure that the installation procedure for your application is consistent with the 386/ix Operating System and to make it easy for your customers to install your package, follow the instructions in this section to make your application installable using `sysadm`. INTERACTIVE delivers the `custom` and `installpkg` commands only to allow you to install existing applications that use these formats.

2.2 The Content and Form of 386/ix Subsets

INTERACTIVE delivers the 386/ix Operating System in packaged *subsets* on one or more diskettes. More than one subset can be put on one diskette, if desired. Tools and utilities are grouped together and packaged as separately installable subsets, so that users may install only the portions of the operating system that are relevant to their needs. Only two subsets are required to create a runtime operating system, Boot and Core. The remainder are optional.

You should be familiar with the format used for all 386/ix subsets so that you can check the software integration process to be sure it has proceeded properly. The contents of a 386/ix subset are stored on one or more diskettes in file systems labelled `install`. Each

file system contains at least two top-level directories, named `install` and *packagename*. The term *packagename* stands for a two-letter mnemonic for the name of the application package. For example, the Kernel Configuration subset is called `kc`. (You will be instructed to name and create this directory for your application in the correct location during the subset integration process. We require that you use a five-letter mnemonic to avoid conflicting names. For example, a word processing package might be called `words`. The other directories will automatically be put in place by the Software Integration Tools.

- *The* `install` *directory*
  The `install` directory must contain two shell scripts named `INSTALL` and `UNINSTALL`. These scripts are automatically called by the 386/ix Operating System when the `installpkg` or `removepkg` option of the `sysadm` utility is run. When you use the tools provided, these shell scripts will be automatically copied into the appropriate directory. If you do not use the tools diskette to integrate your application, you will have to manually type in the scripts given in Appendix B.

- *The packagename directory*
  The other directory must have the same name as the subset, a five-letter mnemonic. Several such directories can be present at this level if more than one subset fits on a single diskette. Each directory should contain three subdirectories: `new`, `install`, and `driver`. A description of the contents of each of these subdirectories follows:

  - *packagename*`/new`
    This directory is the top-level directory for the installed software. When files are copied to the fixed disk, they will be installed in a directory hierarchy that exactly matches the hierarchy under this directory. For example, applications that install a command in `/bin` will store the command on the diskette in *packagename*`/new/bin`.

  - *packagename*`/install`
    This directory contains all the information about the subset, for example, the sequence number of the diskette. It can also contain some optional shell scripts to perform tasks that are typical for your software installation, either before or after loading the files on the fixed disk.

- *packagename*`/driver`
  If the package contains device drivers, a directory for each driver present, containing the driver object and configuration files, will be found in this directory. Otherwise, this directory is empty. See §3 for more information about integrating device drivers.

2.3 Creating an Installable Subset

This section describes how you can prepare your application so that it can be installed as an optional 386/ix subset. Using this procedure will insure that your applications will be installed in a manner that is consistent with other 386/ix applications, making installation easier for the end user.

2.3.1 Before You Begin

Before you can create an optionally installable subset, the following conditions must be met:

- All the files that are part of your package must be installed on your system, either in the final location where the subset will reside on the user's system or in a special directory that acts as the top of your binary tree. Refer to Appendix A if you do not know how to transfer your files onto a 386/ix system.
- The 386/ix Software Development System must be installed on the system where you create the subsets (so that the UNIX `make` utility can be accessed).
- The INTERACTIVE Software Integration Tools subset must be installed on the system.
- You must be logged in to the system as `root`.

2.3.2 Using the Tools to Build the Files for a 386/ix Subset

1. To build your subset, after logging in as `root`, change to the directory `/usr/subsets` and type in the command `setpath`:

   ```
   # cd /usr/subsets
   # . ./setpath
   ```

   This calls a shell script that causes the directory containing the tools to be specified in your `PATH` shell variable.

2. Change to the subdirectory of `/usr/subsets` named `scripts` and create a directory with a five-letter name for

your subset, e.g., `mysub`, in which you will store all the information about the subset you supply:

```
# cd scripts
# mkdir mysub
```

☛ Note that you *must* use a *five* letter name. The integration will fail if you do not follow this convention.

3. Change your directory to `mysub` and create four files with the names `FILES`, `NAME`, `VERSION`, and `copyright`. The contents of these files are described below:

   - `FILES`
     `FILES` should contain a list of all the files that are part of your package, listed with full path names and sorted in reverse alphabetical order. Linked file names should appear as required; they will automatically be linked correctly.

   - `NAME`
     `NAME` should contain a one-line description of the subset, for example:

     ```
     MYSOFT software for cemetary management
     ```

   - `VERSION`
     `VERSION` should contain the latest version number of the package, for example:

     ```
     1.3
     ```

   - `copyright`
     `copyright` should contain a script that echoes copyright information, for example:

     ```
     echo Copyright MYSUB software company 1989
     ```

   The following files in `/usr/subsets/scripts/mysub` are optional; create them if needed:

   - `setup`
     This file can be used for any necessary installation tasks other than linking files (links are automated). This file is automatically executed after the files are copied.

   - `unsetup`
     This file is used to do any necessary removal tasks other than removing links (link removal is automated).

- `preinstall`
  This file is executed before the files are copied. It can be used, for example, to save any files that are going to be overwritten.
- `postinstall`
  This file is executed after the files are copied and the `setup` file has been executed (if present).
- `dependencies`
  Some subsets depend on the presence of others. This file is used to check for the presence of subsets required for the current installation.

4. After you have created the necessary files in the `mysub` directory, you must set two shell variables at the system prompt: `ROOT` and `FLOP`. `ROOT` should be assigned the path name of the top-level directory of your software on the computer where you make the subsets. `FLOP` is used to specify the type of diskettes (5¼" or 3½") you will use. For example, you might use the following command line:

   ```
   # ROOT=/distrib ; export ROOT ; FLOP=3 ; export FLOP
   ```

   The defaults are / (the root directory) and 5, for 5¼" 1.2 MB diskettes. Type `FLOP=3` if you are using 1.44 MB 3½" diskettes.

5. Change to the `/usr/subsets/scripts` directory and type the following command:

   ```
   # mkmkfiles mysub
   ```

   This creates a file called `makefile` in the `mysub` directory.

6. Change to the `mysub` directory. You will now use the `make` command to make your installable subset. The `make` command has several arguments. If you use the argument `floppy`, then the entire procedure will be performed; if you use the argument `proto`, the entire procedure except the actual making of the diskettes will be performed. In practice, we recommend that you make sure all necessary files are created *before* you actually make the diskettes. Type `make proto`.

   ```
   # cd mysub
   # make proto
   ```

The tools on the tools diskette are now invoked to create the files needed to build your application diskette(s). After a period of time (which will vary greatly depending on the complexity of your software), the necessary prototype files will be created in this directory.

2.3.3 What the Tools Have Done Automatically

The Software Integration Tools automate software integration with the 386/ix Operating System. After you have used the command `make proto` to create the prototype files for your subset, you may want to look at the files created in the directory. The tools have performed the following tasks:

- *Built a file list.*
  Using the `FILES` file you have created, the `$ROOT` directory is examined to verify that all files are present. For all the links that are found, the appropriate link commands are stored in a file named `link` and the necessary `rm` commands in a file named `unlink`. All real files are listed in the file `FILELIST.mysub`.

- *Made a* `subset.name` *file.*
  A file that becomes part of the subset and is stored as `/usr/options/`*packagename*`.name` (in our example, `/usr/options/mysub.name`) is created using the information in the files `NAME` and `VERSION`. This file is used by `sysadm` to verify whether a subset has already been installed and whether or not it is the same release.

- *Modified comment sections.*
  386/ix executables are COFF (common object file formats) files. One section of such a file is called the `comment section`, and its contents can be examined using the `what` utility. In older versions of UNIX one could only handle this by defining strings in the source program that were used for no purpose other than displaying them when a `what` command was run on the executable.

  Now there is a utility, called `mcs`, that allows you to modify the comment section of a program *without recompiling*. The 386/ix tools automatically remove the comment sections of all executable programs in the subset and add one line in the format:

```
mysub:file 386/ix 2.0 - Version 1.7
```

where `1.7` is the version number specified in the `VERSION` file, `file` is the name of the file, and `mysub` is the name of the subset.

- *Determined the number of diskettes needed.*
  The next step is to determine how many diskettes will be needed. The sizes of all the files in the subsets are examined, and the list of files is split into groups that will fit on one diskette. The file `NBRDISKS` is created, as well as files called `FILE.`*n* and `SIZE.`*n* for each diskette, where *n* is the diskette number.

- *Built a list of files to be removed.*
  When a user decides to uninstall a package, the `UNINSTALL` program needs to know which files are to be removed. To do this, a file called `Rlist.mysub` is created.

- *Created prototypes.*
  For each diskette needed, a directory called `disk.`*n* is created. All the files that need to go on the diskette, as well as a file called `proto.`*n*, are automatically copied into this directory. `proto.`*n* is a prototype file that can be passed as an argument to the `mkfs` command when a file system is made. For a description of the layout of such a prototype file, refer to *mkfs*(1M) in the *User's/System Administrator's Reference Manual*. If a subset contains many small files, the number of inodes to be specified for the `mkfs` command is also adjusted.

2.3.4 Creating the Master Diskettes

1. Now that all the necessary files are in position, make sure you are in the `mysub` directory and type the command `make floppy` to make the master diskette(s). The system will automatically prompt you to insert the necessary diskette or diskettes.

   ```
   # make floppy
   ```

   The `mkfs` utility then automatically organizes the diskette as a 386/ix file system. You can abort the procedure at this point by typing `s` and using **RETURN**.

2. If you want to continue, insert a formatted diskette and use **RETURN**, or insert an unformatted diskette, type `f`, and then use **RETURN**. It writes some information to the screen, including:

```
(DEL) if wrong
```

Ignore this message. `mkfs` will wait for about 10 seconds and then continue.

3. After making the file system, all necessary files will be copied to the diskette. If your software does not fit onto one diskette, you will be prompted for a second, a third, and so on.

☛ Note that when you type `make floppy`, a dummy file named `floppy` is created in your working directory. To make additional master copies of your software, remove the `floppy` file, then type `make floppy` again.

The diskette or diskettes you have just created are the masters for your new subset. File links will have been created automatically, and your diskettes can be installed using the `sysadm` utility. Test your new subset to verify that all software has been installed and setup has been completed.

3. INTEGRATING DEVICE DRIVERS WITH 386/ix SUBSETS

3.1 386/ix and the UNIX Kernel

Many UNIX implementations come with a single UNIX kernel that cannot be modified. While this may be reasonable for a mainframe computer, it is completely unworkable for PC-based UNIX systems. Configurations range from 2 MB to 16 or more MB of physical memory, and several peripherals can be installed that require a device driver to be added to the kernel. Users with PC-based machines need to be able to change tunable parameters and add or remove drivers in order to build a UNIX kernel that fits their configuration and needs.

Because the 386/ix Operating System contains the objects of the UNIX kernel and its related device drivers, it can be reconfigured by users to suit their individual requirements. On some PC-based UNIX systems, the 386/ix Software Development System must be installed in order to build a new kernel. This is not true of 386/ix. The user can build a new kernel with only the basic 386/ix Operating System installed.

3.2 The `kconfig` Utility

`kconfig` is one of INTERACTIVE's major enhancements to the 386/ix Operating System. It is an easy-to-use menu-driven interface distributed as part of the basic 386/ix Operating System. In conjunction with the utilities provided in the Kernel Configuration subset, this tool allows even a naive UNIX user to configure, build, and install a customized 386/ix kernel. We urge you to make sure that your driver(s) can be installed using this program and that your documentation explicitly states that the 386/ix Software Development System does not need to be installed to add a driver to the 386/ix kernel. Only the Kernel Configuration subset is required. Refer to the 386/ix manual entry *kconfig*(1) in Appendix D for more information about this utility.

3.2.1 `kconfig` *and Related AT&T Utilities*

For the sake of compatibility, 386/ix Release 2.0 uses the AT&T UNIX System V/386 Release 3.2 utilities and file formats for installing drivers. These utilities are transparent to the user, since they are automatically called by `kconfig`. The name of each AT&T utility begins with the letters "id," so they are sometimes referred to as the **id** utilities (**i**nstallable **d**rivers). These utilities and their interface are described in the *Integrated Software Development*

Guide that is delivered with the 386/ix Software Development System. It contains information about how to write a UNIX device driver and how to use the id commands. It does not refer to `kconfig` or other INTERACTIVE enhancements, however, so we advise you to use that document in conjunction with the information presented here.

3.3 Making a Device Driver Usable on the 386/ix Operating System

Once a device driver (and its related software) has been developed and tested, it must go through three steps before it is actually operational and can be used with the peripheral for which it is intended. It must be:

- Packaged for distribution, that is, made into a 386/ix subset.
- Installed on the customer's system and prepared for installation by the customer.
- Configured into the kernel and the new kernel successfully installed.

The following sections illustrate the components necessary for generating a valid driver package and the additional tasks that need to be performed to build a 386/ix subset that contains one or more device drivers. For more details, refer to the *Integrated Software Development Guide*.

3.4 The Environment of a 386/ix Device Driver

A device driver is not a standalone piece of software. For example, as discussed in Appendix A, a diskette driver requires a number of *special files* that serve as the link between the kernel and the file system. When a utility attempts to access a file that is a special file, the system will determine which driver code needs to be executed by examining the *major device number* of this special file.

For some drivers, especially those that involve terminals, such as the serial port driver or any multiport board driver, additional configuration procedures are necessary to make them work. Entries need to be added to a system file, called `/etc/inittab`, so that a logger is activated on these particular ports as soon as the system enters multi-user mode (init levels 2 or higher). For example, the typical `/etc/inittab` entry below would activate a terminal connected to the first port of a Hub6™ card, for which `/dev/tty4a` is the name of the special file.

```
4a:23:respawn:/etc/getty tty4a 9600
```

On the 386/ix Operating System, `/etc/inittab` entries are modified using `sysadm ttymgmt`.

Previous releases of 386/ix required that:

- All special files had to be present on the system, even if the associated drivers were not in use.
- All loaded drivers had major device numbers that were pre-assigned by their developer.
- All possible entries were listed in the `/etc/inittab` file.

These requirements caused various problems. For example, two independent vendors could develop drivers with the same pre-assigned major device number. This was not a problem unless a user attempted to configure both drivers into the same kernel. Some vendors developed smart scripts to examine the entire system before assigning the major device number. In Release 2.0 of the 386/ix Operating System, this is no longer necessary; the `id` utilities automatically assign an unused major device number.

Different drivers can now use the same names for special files without causing any system problems. Beginning with Release 2.0 of the 386/ix Operating System, all special files are recreated whenever a new kernel is installed, and the unnecessary ones are removed. The `/etc/inittab` file is recreated containing only the entries related to the installed drivers. Thus, the number of possible conflicts between different device drivers has been reduced tremendously.

☛ Note that some hardware-related limitations still exist. Features such as interrupt vectors still need to be pre-assigned and, in the case of a conflict, manually changed.

3.5 Preparing Your Driver for Installation

To take advantage of the features of the new device driver environment, a driver needs to be prepared for installation. `insdriver` is the 386/ix utility that handles this process. One of the tasks it performs is to add a one-line description of a driver to the file where such information is stored. All loaded driver modules are subsequently advertised in the `kconfig` menus with this one-line description of their function. Refer to the *insdriver*(1) manual

entry in Appendix D for more information about the `insdriver` utility and its function.

To properly prepare your driver for installation, you must have the following files (some of which are optional) in a single directory. An example will be given later.

- `Driver.o`
  This is the driver module that is to be configured into the kernel. In previous releases of the 386/ix Operating System, these modules had the same name as the driver, for example, `hub.o`. Now each driver module always has the same name, `Driver.o`.

- `Master`
  This file contains on-line information about the driver. The driver name, the functions it contains, and so on, are specified in it. Refer to *mdevice*(4) in the *Programmer's Reference Manual* for details about this file. The block and/or character device major number specified should be listed as zero.

- `System`
  This one-line file contains the name of the driver, which should match the one in the `Master` file, an `N` or `Y` to indicate whether or not the driver should be installed (note that it should always be `N` at this point), and some hardware-specific characteristics. Refer to *sdevice*(4) in the *Programmer's Reference Manual* for details about this file.

- `Space.c` (optional)
  This optional source file contains driver-specific declarations of data structures.

- `Node` (optional)
  This file specifies the special files the driver will need to operate, one line for each. The format of such a line is:

  ```
  drivername devicename X minor
  ```

  `X` is replaced by a `c` for character special files or a `b` for block special files. Note that the major device number need not be specified. It still needs to be assigned at this point. Making consistent use of the driver name is the important point here.

- `Init` (optional)
  This file should contain all `/etc/inittab` entries required for this driver to work properly.

3.5.1 Sample `Master`, `System`, `Init`, *and* `Node` *Files*

Refer to *mdevice*(4) and *sdevice*(4) in the *Programmer's Reference Manual* for the proper description of the meanings of all fields in these files.

- `Master`

```
taco  Iocrwi  iHct   taco   0    0    6    24    -1
```

The driver's name is `taco`, it has an `init`, `open`, `close`, `read`, `write`, and `ioctl` routine, is for hardware and installable, is a terminal device, and uses character special files. The major device number is set to 0. The driver assumes a minimum of 6 subsystems and a maximum of 24 and does not use DMA.

- `System`

```
taco   N    6    7    2    3    302   308   0    0
```

The driver named `taco` is not installed by default, uses 6 sub-devices, runs at *ipl* level 7, requires a sharable interrupt line, uses interrupt vector 3, and has 302 and 308 as start and end I/O addresses.

- `Init`

```
taco0:23:off:/etc/getty ttyt0 9600
taco1:23:off:/etc/getty ttyt1 9600
taco2:23:off:/etc/getty ttyt2 9600
taco3:23:off:/etc/getty ttyt3 9600
taco4:23:off:/etc/getty ttyt4 9600
taco5:23:off:/etc/getty ttyt5 9600
```

These are the entries that are added to the newly created `/etc/inittab` file when a kernel containing this driver is installed. `sysadm ttymgmt` is used to activate the `getty` commands.

- `Node`

```
taco     ttyt0   c     0
taco     ttyt1   c     1
taco     ttyt2   c     2
taco     ttyt3   c     3
taco     ttyt4   c     4
taco     ttyt5   c     5
```

These nodes will be created in `/dev` using the minor device numbers specified and the major device number that was assigned automatically. Subdirectories of `/dev` can be specified if needed.

3.6 Preparing Your Driver Using `insdriver`

1. After all the files are in place, make sure you are in the same directory as the files and type `insdriver`. Your screen will look similar to this:

   ```
   Directory Containing Driver Files:
   ```

2. Answer `.` for the current directory, followed by RETURN. Your screen will then look similar to this:

   ```
   Driver Name:
   ```

3. Release 2.0 device driver modules are located in subdirectories of the directory `/etc/conf/pack.d`. In the future, INTERACTIVE may assign names for these directories to third-party software developers, to avoid naming conflicts. At present we recommend using names of four or five characters rather than three characters. Type in the driver name and use RETURN. Your screen will then look similar to this:

   ```
   Configuration Directory (/etc/conf):
   ```

4. Use RETURN. Your screen will then look similar to this:

   ```
   Enter a one line driver description:
   ```

5. Enter a comprehensive description of what the driver is meant to be used for, followed by RETURN. Your screen will then look similar to this:

   ```
   Do you want to Configure/Build a Kernel [ y , n ] :
   ```

6. If you have no more drivers to prepare, type `y`; otherwise, type `n`.

At this point, the `insdriver` program does the following:

- Adds the one-line description you specified to the file `/etc/conf/kconfig.d/description`.
- Moves the `Node` file to `/etc/conf/node.d`, giving it the name of your driver.
- Moves the `Init` file to `/etc/conf/init.d`, giving it the name of your driver.
- Edits the file `/etc/conf/cf.d/mdevice` to add information about your driver, including a major device number.
- Edits the file `/etc/conf/cf.d/sdevice` to add information about your driver.

The driver is now ready for integration into the kernel. Release 2.0 of the 386/ix Operating System uses an entire directory (`/etc/conf/cf.d`) to describe the kernel configuration, rather than a single file (`/etc/atconf/systems/system.std`), as in previous releases. When configuring the kernel using `kconfig`, backup copies of all these files are made in a subdirectory named `OLD`, and subsequently all files are modified according to the choices you specified while using the `kconfig` menu. When a new kernel is installed, the following actions are taken automatically:

- The new kernel is moved to `/unix`, the current one to `/OLD.unix`.
- A new `/etc/inittab` file is created, by appending all files that reside in `/etc/conf/init.d` to the contents of `/etc/conf/cf.d/init.base`, except for those files that have the same name as a driver that is not configured.
- For all files found in `/etc/conf/node.d`, the special files decribed there are removed for all drivers not configured, and recreated for those that are.

As a result, a minimum size `/etc/inittab` file and only those special files that are actually used by the kernel are stored on your 386/ix system.

3.7 Using the Tools to Build a 386/ix Subset Containing Device Drivers

The same tools that were described in an earlier section can be used to make a 386/ix subset that also contains one or more device drivers. This section explains the additional tasks that need to be performed to do this.

1. Create a directory with a five-letter name in `/usr/subsets/scripts` exactly as described before:

```
$ cd /usr/subsets
$ . ./setpath
$ cd scripts
$ mkdir mysub
```

2. Determine at this point what the value of the `ROOT` variable will be. Create a directory named `$ROOT/etc/drivers` (unless it already exists) and a subdirectory for each device driver that goes into the subset. In each of these directories, the files `Driver.o`, `Master`, and `System` and other

optional files belonging to that driver should be stored. The tools will understand that these files are driver files because of their special location.

3. Now create all necessary files (as mentioned in §2.3.3) and make sure that in `FILES` you list the location of all driver files and directories as `/etc/drivers` (i.e., do not use the prefix `$ROOT`).

4. Add to the now mandatory `setup` file a script that will add a line to `/etc/conf/kconfig.d/description` for each driver in the subset. A typical description line is:

```
name  -     -     name  -     description of driver
```

   A sample setup script that will do this is included in Appendix B.

   You should also include a script in your driver that instructs the user to make a new kernel using the `kconfig` utility when your driver software is loaded. Every subset containing drivers should contain a `setup` or `postinstall` shell script that:

   - *Adds a description to the description file.*
     The first time the package is installed, a description of the driver should be added to `/etc/conf/kconfig.d/description`. If it is a new release, the description should either be replaced or left unchanged. Use the `grep` command to find out. (Refer to the *grep*(1) manual entry in the *User's/System Administrator's Reference Manual* for information about the `grep` command.)

   - *Calls* `kconfig`.
     When the software is loaded onto the fixed disk, either query the user or call the `kconfig` utility immediately.

     It is possible to automate the user's interaction with `kconfig`, although this is optional. A sample `postinstall` file may be found on the tools diskette and in Appendix B.

5. Perform all other steps required to create a regular 386/ix subset, as described in §2.3.3.

3.7.1 What the Tools Do for You

The Software Integration Tools perform the following:

- They examine the `FILES` file and will, for each directory listed in `/etc/drivers`, store it on the diskette as a subdirectory of *packagename*`/driver`, instead of *packagename*`/new`. When the subset is installed, the `idinstall` (not `insdriver`) command will be automatically executed and the driver modules installed in the correct places ready for kernel configuration, and a major device number will be assigned for each of them. The script in the `setup` file is required to properly interface with the `kconfig` utility.
- They generate a file called `drivers` and store it as *packagename*`/install/drivers`. This contains a list of all the driver directories, so `sysadm` will remove them using the `idinstall` utility, when needed.

4. UPDATING 386/ix DEVICE DRIVERS FROM RELEASE 1.0.6

4.1 Comparing 386/ix Releases 2.0 and 1.0.6

Release 2.0 of the 386/ix Operating System is upwardly compatible with respect to 386/ix Release 1.0.6. This means that all utilities and applications that run on 1.0.6 will run on 2.0 without recompiling. Release 2.0 allows you to run existing XENIX binaries as well. If you are also using the VP/ix™ package, you have a system that can run:

- MS-DOS® executables
- XENIX executables
- UNIX SYSTEM V/286 executables
- UNIX SYSTEM V/386 executables, regardless of whether they are built on a V.3.0 (386/ix 1.0.6), V.3.1, or V.3.2 system

The UNIX kernel, however, has undergone many changes and contains many enhancements. A device driver, when installed, is part of the kernel and needs to reside with this new kernel. Therefore, many existing device drivers will need to be recompiled, and for some of them the source code will need to be slightly modified. The `tty` structure, for example, has been modified, and as a result all drivers for multiport cards will at lease need to be recompiled.

One could take an existing 386/ix 1.0.6 driver object, create the appropriate `Master` and `System` files for it, and subsequently succesfully generate and boot a 2.0 kernel with it. Even then, however, the driver might not work properly.

The following section describes guidelines for converting these drivers. These guidelines are intended for driver developers, not customers who have a driver object from a third-party vendor.

☛ We recommend that end-user customers contact hardware vendors directly for updated versions of drivers that are not bundled with the 386/ix Operating System.

4.2 Converting Drivers from 1.0.6 to 2.0

Your 1.0.6 device driver modules reside either on your fixed disk or in a 386/ix 1.0.6 subset in subdirectory *name* of `/etc/atconf/modules`. These files typically include *name.o*, `config`, and `space.c`. This section describes what you need to do to convert your 1.0.6 driver files to 2.0 driver files.

1. Create a directory somewhere on the system and move *name.o* to that directory and call it `Driver.o`. If `space.c` is present, move this file also and name it `Space.c`.
2. Create the new `Master` and `System` files there and edit them so that *name* begins in the first column of the first field and `N` is second field in the `System` file. Each file should contain one line, with different fields separated by spaces or tabs.
3. Examine the contents of the `config` and original `space.c` files. Here are some typical occurrences in these files and the corresponding information needed in the `Master` and `System` files:

   - `block(69)`
     This is a block device. Add a letter `b` to the third field of the `Master` file and make sure the letters `i` and `H` appear there. Put a zero (`0`), not `69` as the fifth field.
   - `character(69)`
     This is a character device. Add a letter `c` to the third field of the `Master` file and make sure the letters `i` and `H` appear there. Put a zero (`0`), not `69` as the sixth field.

- `prefix =` *nam*
  This is the prefix beginning all driver routine names. List it as the fourth field of the `Master` file.
- `intvec =` *3*
  This is the interrupt vector used by the driver. Make it the sixth field of the `System` file.
- `intpri = SPL` *5*
  This is the `ipl` level expected by the driver. Make it the fourth field of the `System` file.
- `functions = open, close ...`
  These are the routines the driver supports. For each of them, add a letter in the second field of the `Master` file, according to the following rule: `o` for open, `c` for close, `r` for read, and so on, as described in *mdevice*(4) in the *Programmer's Reference Manual*.
- *Other*
  Other characteristics, such as DMA channel and starting address, may appear. Fill out the appropriate fields; otherwise use `0` (`-1` for DMA channel). Note that if a starting address is specified, an ending address should be specified as well.

4. Now run `insdriver` as specified in §3.6 and build a new kernel to test the behavior of the driver. Whether it will work depends on whether or not some of the driver characteristics were hard coded in the original module. Refer to your hardware documentation for additional information on starting addresses, DMA channels, and so on, if your first attempt is not successful.

Appendix A: USING DISKETTES WITH 386/ix

Diskettes are the easiest medium for quickly storing files and moving them from place to place on the 386/ix Operating System. You may want to use diskettes to move files you already have on another computer system onto the 386/ix system before you begin to integrate your application or driver. This appendix is provided as a convenience for developers who are new to UNIX systems. It provides a quick overview of the UNIX-specific utilities and information you need to use diskettes with the 386/ix Operating System.

Refer to section 8.3, "Device Naming Conventions," and section 11.12, "Adding a Second Diskette Drive," in "386/ix Maintenance Procedures" for more information on the topics covered in this appendix. For additional information about UNIX operating systems in general, refer to the "UNIX Primer" that accompanied your 386/ix Operating System, and the *User's Guide* and *User's/System Administrator's Reference Manual*.

1. DEVICES, DEVICE SPECIAL FILES, AND DISKETTES

Devices may be treated by UNIX systems as either *block* devices or *character* devices. Block devices, such as fixed disks, process information in blocks. Character devices, such as terminals, process information one character at a time.

The 386/ix Operating System supports several diskette formats and can treat a diskette as either a block device or a character device. The two most commonly used diskette sizes are 5¼" diskettes and 3½" diskettes. Different densities are also available, ranging from 360K to 1.44 MB.

All devices on the 386/ix Operating System, including diskettes, are associated with a *special file*. All special files that refer to block devices are stored in the directory `/dev/dsk`. All special files that refer to character devices are stored in the directory `/dev/rdsk`. Each different diskette configuration that is supported on your system must be represented on the 386/ix Operating System by its own corresponding special file. A number of these special files are supplied with the 386/ix Operating System. Special file names for diskettes are rather cryptic in the UNIX system; for example, the special file name for a 5¼", 1.2 MB diskette is `f0q15dt`. The names are based on the following factors:

- Position of the diskette drive (first – `f0` or second – `f1`)
- Size of the diskette (5¼" – `9` or `15` sectors per track or 3½" – `9` or `18`)
- Density of the diskette (single or double –`d` or quad – `q`)
- Use of the whole diskette (`dt`) or skip the first cylinder, as on the *Boot* diskette (`d`)

For the sake of convenience, you can link a more comprehensible name, such as `/dev/floppy` to a special file, using the UNIX `ln` command. For example, you could log in as `root` and type:

```
ln /dev/rdsk/f0q15dt /dev/floppy
```

From then on you can simply use `/dev/floppy` in command lines requiring your diskette device special file name. This is easier to remember and simplifies typing.

2. MAKING A DEVICE SPECIAL FILE

The 386/ix Operating System is delivered with device special files for the most commonly used diskettes already in the proper directories. However, if the type of diskette that you want to use does not have the appropriate device special file already on your system, you can create the necessary file using the UNIX `mknod` command. The general form of the command is shown in the following table:

Command Name	*File Name*	*Type of Device*	*Major Device Number*	*Minor Device Number*
`mknod`	*name*	`c`	`1`	*minor no.*

For example, the command

```
mknod /dev/rdsk/f1q18d  b  1  81
```

makes a device special file called `/dev/rdsk/f1q18d` that will be treated as a block device, has the major device number `1` and a minor device number of `81`. (The major device number for diskette drives is always `1` on the 386/ix Operating System.) Use the letter `c` instead of the letter `b` and place the file in `/dev/dsk` rather than `/dev/rdsk` if you want the device to be treated as a character device.

The following table lists some of the minor device numbers you can use and the meaning of each. The file names are those used in the

386/ix Operating System. (Names ending in `dt` indicate the names used for the entire diskette; those ending in `d` indicate the same type of diskette minus the first cylinder.)

Device Name	*Minor*	*Description*
f0d9dt	16	360K, 51/4" disk, first diskette drive
f0d9d	20	
f0q9dt	96	720K, 31/2" disk, first diskette drive
f0q9d	100	
f0q15d	0	1.2MB, 51/4" disk, first diskette drive
f0q15d	4	
f0q18d	80	1.44MB, 31/4" disk, first diskette drive
f0q18d	84	
f1d9dt	17	360K, 51/4" disk, second diskette drive
f1d9d	21	
f1q9dt	97	720K, 31/2" disk, second diskette drive
f1q9d	10	
f1q15d	1	1.2MB, 51/4" disk, second diskette drive
f1q15d	5	
f1q18d	81	1.44MB, 31/4" disk, second diskette drive
f1q18d	85	

3. COPYING FILES ONTO A DISKETTE

Once you have verified that the appropriate device special file is present for the type of diskette you want to use or you have created the appropriate file, you are ready to copy your files onto a 386/ix diskette. The UNIX `cpio` and `tar` commands can be used in combination with the appropriate device special file to copy your files onto a 386/ix diskette. (Remember that if you are using a diskette for the first time, it must be formatted.)

Either of the following commands can be used to copy all the files that are found in the current directory onto a diskette:

```
ls | cpio -oBcvdu > /dev/rdsk/f0q15dt
```

or

```
tar cvf /dev/rdsk/f0q15dt .
```

If the files to be saved will require more than one diskette, use `cpio` rather than `tar`. With `cpio`, you can change diskettes without having to interrupt the program. Refer to the *cpio*(1) and

tar(1) manual entries in the *User's/System Administrator's Reference Manual* for more information about these utilities.

4. MOUNTING DISKETTE FILES

As mentioned previously, `/dev/rdsk` is a directory containing various special files representing character devices. `/dev/dsk` contains the equivalent block devices for diskettes. You will need these special files when you want to use diskettes as mountable file systems. (386/ix subsets are mountable file systems.) 386/ix subsets are initialized using the `mkfs` utility and from then on can be mounted using the `mount` command:

```
mount /dev/dsk/f0q15dt  /mnt
```

After mounting, all files on the diskette can be examined by exploring the directory `/mnt` (any directory name could be used as a mounting point). Although the use of diskettes in this fashion may cause some overhead, it is probably the most convenient way to add files to diskettes that already contain some files.

Appendix B: SAMPLE SHELL SCRIPTS

1. install

```
#
#  Copyrighted as an unpublished work.
#  (c) Copyright 1987 INTERACTIVE Systems Corporation
#  All rights reserved.
#
#  RESTRICTED RIGHTS
#
#  These programs are supplied under a license.  They may be
#  used disclosed, and/or copied only as permitted under such
#  license agreement.  Any copy must contain the above copyright
#  notice and this restricted rights notice.  Use, copying,
#  and/or disclosure of the programs is strictly prohibited
#  unless otherwise provided in the license agreement.
#
set -v
#ident  "@(#)INSTALL.sh 2.5 - 88/10/06"
#
#       find mount device
#
if      test -b "$1"
then
        mntdev=$1
elif    test -b /dev/diskette
then
        mntdev=/dev/diskette
elif    test -b /dev/install
then
        mntdev=/dev/install
else
        echo "**ERROR** Can't find mount device"
        exit
fi
mntname="${2:-/install}"
#
#       Get the packages on this diskette
#
INSDIR=/usr/lib/installed
INSPATH=/install
CONF=/etc/conf
cwd=`pwd`
cd $INSPATH
set `ls -d ?????`
cd $cwd
nbrpkgs=$#
pkgs="$*"
for pkg in $pkgs
do
        NAMEFILE=/install/$pkg/install/$pkg.name
        PKGNAME=`cat $NAMEFILE`
        echo Install the ${PKGNAME:-$pkg}? "(y): \c"
        read resp
        [ "$resp" = "n" -o "$resp" = "N" ] && continue
        INSPATH=/install/$pkg/new
        DRIVER=/install/$pkg/driver
        export PKGNAME
        LAST=`cat /install/$pkg/install/NBRDISKS`
        echo "Installing the ${PKGNAME}."
        [ -x /install/$pkg/install/copyright ] \
```

```
&& /bin/sh /install/$pkg/install/copyright

expect=1
while   [ "$expect" -le "$LAST" ]
do

    if      [ "$expect" -gt 1 ]
    then
            cd /
            /etc/umount $mntdev 2>/dev/null
            echo "Remove floppy and insert floppy"
            echo "number $expect"
            echo "Type <return> when ready: \c"
            read answer
            /etc/mount $mntdev $mntname -r 2>/dev/null
    else
            if [ "$expect" -eq 1 -a \
               -r /usr/options/$pkg.name ]
            then
                echo "**WARNING**       The \
                      `cat /usr/options/$pkg.name`\
                      is already installed"
                answer=""
                while [ "$answer" != y -a "$answer" != n ]
                do
                    echo "Type 'y' to overwrite the \c"
                    echo "${PKGNAME} or 'n' quit: \c"
                    read answer
                done
                case $answer in
                    y) ;;
                    n)      cd /
                            /etc/umount $mntdev 2>/dev/null
                            exit 1;;
                esac
            fi
    fi

    #
    # Check to make sure that this floppy belongs to the
    # package
    #

    while   [ ! -s $NAMEFILE -o "${PKGNAME}" \
            != "`cat $NAMEFILE`" ]
    do
        echo "**ERROR** Floppy does not belong to"
        echo "the ${PKGNAME}"
        cd /
        /etc/umount $mntdev 2>/dev/null
        echo "Remove floppy and insert correct floppy"
        echo "Type <return> when ready: \c"
        read answer
        /etc/mount $mntdev $mntname -r 2>/dev/null
    done
```

```
#
# Check to make sure this is the correct floppy of the
# set
#

answer=""
while   [ "`cat /install/$pkg/install/ORDER`"\
        != "$expect" -a "$answer" != y ]
do
    echo "**WARNING**       Floppy out of sequence"
    echo "Expecting floppy diskette number $expect"
    while [ "$answer" != y -a "$answer" != n ]
    do
        echo "Type 'y' to continue or \c"
        echo "'n' to try another floppy: \c"
        read answer
    done
    case $answer in
    n)      cd /
        /etc/umount $mntdev 2>/dev/null
        echo "Remove floppy and insert correct floppy"
        echo "Type <return> when ready: \c"
        read answer
        answer=""
        /etc/mount $mntdev $mntname -r 2>/dev/null
        ;;
    y)      expect=`cat /install/$pkg/install/ORDER`
        ;;
    esac
done

#
#       Verify dependencies loaded
#

#
# Verify that there is space for the package.
#
if [ "$expect" -eq 1 -a -x \
   /install/$pkg/install/dependencies ]
then
    if /bin/sh /install/$pkg/install/dependencies
    then :
    else
            exit 1
    fi
fi

if /etc/mount | grep /usr >/dev/null 2>&1
then

    USRneeds=`expr "\`du -s ${INSPATH}/usr\`"\
              : "\([0-9]*\).*"`
    USRspace=`expr "\`df /usr 2>/dev/null\`"\
              : '.*: *\([0-9]*\)'`
```

```
        if
            [ "$USRspace" -lt "$USRneeds" ]
        then
            echo "**ERROR**"
            echo "${PKGNAME} cannot be installed --"
            echo "Not enough space on the hard disk."
            echo "There are $USRspace blocks available"
            echo "on the /usr file system --"
            echo "$USRneeds blocks are needed."
            exit
        fi
    else
        USRneeds=0
    fi
    if expr `ls -a ${DRIVER} | wc -l` \> 2 >/dev/null
    then
        ROOTneeds=`expr "\`du -s ${DRIVE}\`" \
        : "\([0-9]*\).*"`
        drexist=1
    else
        ROOTneeds=0
        drexist=0
    fi
    ROOTneeds=`expr "\`du -s ${INSPATH}\`" \
              : "\([0-9]*\).*"`
    ROOTneeds=`expr ${ROOTneeds} - ${USRneeds}`


    ROOTspace=`expr "\`df / 2>/dev/null\`" \
              : '.*: *\([0-9]*\)'`


    if
        [ "${ROOTspace:-2000}" -lt "$ROOTneeds" ]
    then
        echo "**ERROR**   ${PKGNAME} cannot be installed --"
        echo "Not enough space on the hard disk."
        echo "There are $ROOTspace blocks available"
        echo "on the / (root) file system --"
        echo "$ROOTneeds blocks are needed."
        exit
    fi

    #       Do special work before files are copied in.
trap "trap '' 1 2 3 9 15; rm -f /tmp/*$$; exit 1" 1 2 3
    if [ "$expect" -eq 1 ]
    then
        cp /install/$pkg/install/Rlist.$pkg /tmp/Files$$
        cp /install/$pkg/install/Remove /tmp/Remove$$
        if [ -f /install/$pkg/install/preinstall ]
        then    /bin/sh /install/$pkg/install/preinstall
        fi
    fi
    if [ "$drexist" = 1 ]
    then
        cd ${DRIVER}
        echo "Installing driver(s):  please wait"
        for dir in `ls`
        do
            cd $dir
            ${CONF}/bin/idcheck -p $dir > /dev/null 2>&1
            if [ $? != 0 ]
            then
```

```
                    if [ -d ${CONF}/pack.d/$dir ]
                    then
                        ${CONF}/bin/idinstall -u -k $dir
                    else
                        ${CONF}/bin/idinstall -a -k $dir
                    fi
                else
                    ${CONF}/bin/idinstall -a -k $dir \
                        >/dev/null 2>&1
                fi
                if [ $? != 0 ]
                then
                    echo "\n\tThe installation cannot be"
                    echo "completed due to an error in the"
                    echo "'$dir' driver installation."
                    exit 1
                fi
                cd ..
            done
        fi

        if expr `ls -a ${INSPATH} | wc -l` \> 2 >/dev/null
        then
            cd ${INSPATH}
            echo "The following files are being installed:"
            find . -print | cpio -pduvm /
        fi

        #       Do special work after files are copied in.
        cd /install/$pkg/install
        if [ -f link ]
        then
            . ./link
        fi
        if [ -f setup ]
        then
            . ./setup
        fi
        echo "Floppy diskette number $expect is complete"
        expect=`expr $expect + 1`
done

#       Do special work after files are copied in.
cd /install/$pkg/install
if [ -f postinstall ]
then
    . ./postinstall
fi

if [ ! -d /usr/options ]
then
    mkdir /usr/options
    chmod 755 /usr/options
fi
echo "$PKGNAME" >/usr/options/$pkg.name
chmod 744 /usr/options/$pkg.name

if [ ! -d ${INSDIR} ]
then
    mkdir ${INSDIR}
    chmod 755 ${INSDIR}
fi
if [ ! -d  ${INSDIR}/Files ]
```

```
        then
            mkdir ${INSDIR}/Files
            chmod 755 ${INSDIR}/Files
        fi
        mv /tmp/Files$$          ${INSDIR}/Files/$pkg.name

        if [ ! -d ${INSDIR}/Remove ]
        then
            mkdir ${INSDIR}/Remove
            chmod 755 ${INSDIR}/Remove
        fi
        mv /tmp/Remove$$         ${INSDIR}/Remove/$pkg.name
        echo "Installation of the ${PKGNAME} is complete."
        cd $cwd
done
exit
```

2. uninstall

```
#
#  Copyrighted as an unpublished work.
#  (c) Copyright 1987 INTERACTIVE Systems Corporation
#  All rights reserved.
#
#  RESTRICTED RIGHTS
#
#  These programs are supplied under a license.  They may be
#  used disclosed, and/or copied only as permitted under such
#  license agreement.  Any copy must contain the above copyright
#  notice and this restricted rights notice.  Use, copying,
#  and/or disclosure of the programs is strictly prohibited
#  unless otherwise provided in the license agreement.
#

#ident  "@(#)UNINSTALL.sh       2.2 - 88/09/02"
#
#       Get the packages on this diskette
#
CONF=/etc/conf
INSPATH=/install
ERROR="An error was encountered during removing the driver,"
ERROR2="package removal failed"
cwd=`pwd`
cd $INSPATH
set `ls -d ?????`
cd $cwd
nbrpkgs=$#
if [ "$nbrpkgs" -eq "1" ]
then
        pkgs=$1
else
        pkgs="$*"
fi
for pkg in $pkgs
do
    PKGNAME=`cut -f1 -d'-' /install/$pkg/install/$pkg.name`
    FREL=`cut -f2 -d'-' /install/$pkg/install/$pkg.name`
    if [ -s /usr/options/$pkg.name ]
    then
        HREL=`cut -f2 -d'-' /usr/options/$pkg.name`
    else
        HREL=$FREL
    fi

    if [ "$FREL" != "$HREL" ]
    then
        echo "\nThe Version of $PKGNAME on the hard disk"
        echo "is $HREL, which is different than the"
        echo "Version on this floppy ($FREL)"
        echo "The wrong files may be removed."
    fi

    echo Remove the ${PKGNAME:-$pkg} package? "(y): \c"
    read resp
    [ "$resp" = "n" -o "$resp" = "N" ] && continue
    #
    #       Remove drivers before unsetup calls kconfig
    #
    if [ -f /install/$pkg/install/drivers ]
    then
```

```
        echo "The following driver(s) is being removed:"
        cat /install/$pkg/install/drivers | while read dname
        do
            echo "\t$dname"
            ${CONF}/bin/idcheck -p $dname >/dev/null 2>&1
            if [ $? != 0 ]
            then
                if [ -f ${CONF}/pack.d/$dname/stubs.c ]
                then
                    ${CONF}/bin/idinstall -gs $dname | \
sed -e 's/Y/N/' >System
                    ${CONF}/bin/idinstall -eus $dname

                    ${CONF}/bin/idinstall -dopnirhc1z\
                       $dname >/dev/null 2>&1
                else
                    ${CONF}/bin/idinstall -d $dname\
                       >/dev/null 2>&1
                fi
                if [ $? != 0 ]
                then
                    echo $ERROR
                    echo $ERROR2
                    exit 1
                fi
            fi
        done
    fi
    #
    #       Remove linked files and invoke unsetup file
    #
    [ -f /install/$pkg/install/unlink ] \
      && /bin/sh /install/$pkg/install/unlink
    [ -f /install/$pkg/install/unsetup ] \
      && /bin/sh /install/$pkg/install/unsetup
    #
    # Remove the files found in the Rlist file from the hard
    # disk.
    #

    echo "The following files are being removed:"
    for i in `cat /install/$pkg/install/Rlist.$pkg`
    do
        echo $i
        rm -fr $i
    done

    echo "The ${PKGNAME} has been removed."
done
```

3. dependencies

```
# Check that the XXYZZ has already been installed.
#

if
    test ! -s /usr/options/xxyzz.name
then
    echo "**ERROR**    ${PKGNAME} cannot be installed --"
    echo "It requires the XXYZZ Package to be installed first."
    exit 1
fi
```

4. postinstall

```
# Shell script to install driver description
# and then call kconfig
    if fgrep "MYSUB description" \
       /etc/conf/kconfig.d/description > /dev/null 2>&1
    then
    echo Driver information is already there
    /etc/kconfig
    else
    cat <<-\! ¦ ed -s /etc/conf/kconfig.d/description
    $a
    taco    -    -     taco     -             MYSUB description
    .
    w
    q
    !
    /etc/kconfig
    fi
```

Appendix C: BUILDING A SUBSET WITHOUT THE TOOLS

1. BUILDING A SUBSET WITHOUT DRIVERS

The tools on the Software Integration Tools subset diskette are not required to make a 386/ix installable subset. However, if you do not use the tools, you must perform most of the process manually. The necessary steps for a subset consisting of only one diskette are described here.

This section assumes that you have read and understood the concepts in §1 and 2, "INTRODUCTION" and "INTEGRATING APPLICATIONS WITH THE 386/ix OPERATING SYSTEM."

1. Create a file system on the diskette using the `mkfs` utility:

   ```
   /etc/mkfs /dev/dsk/f0q15dt 2300:250 2 30
   ```

   Use the device name `f0q15dt` for 5¼" diskettes; for 3½" diskettes, use the device name `f0q18dt` rather than `f0q15dt`.

2. Label the diskette:

   ```
   /etc/labelit /dev/dsk/f0q15dt install mysub.1
   ```

   where `mysub` is the name of the package and `.1` indicates that it is the first diskette.

3. Mount the diskette by typing:

   ```
   mount /dev/dsk/f0q15dt /install
   ```

4. Create the appropriate directories:

   ```
   mkdir /install/install
   mkdir /install/mysub
   mkdir /install/mysub/driver
   mkdir /install/mysub/install
   mkdir /install/mysub/new
   ```

5. Copy the `INSTALL` and `UNINSTALL` scripts to the `/install/install` directory. If you copy these scripts from another 386/ix subset rather than typing them in manually, make sure you replace all instances of `??` by `?????` in these scripts.

6. In the directory `/install/mysub/install`, create the following files with the following contents:

   - `NBRDISKS`
     This contains the number of diskettes in the subset, typically 1.

- `ORDER`
  This contains the number of this diskette in the subset, typically 1.
- `Rlist.mysub`
  This contains a list of files that need to be removed when the package is removed.
- `mysub.name`
  This contains a one-line description of the subset and its version number.
- `copyright`
  This contains a shell script that will produce copyright information.

The optional files mentioned in §2.3.3, such as `postinstall` and `setup`, can be installed in this directory as well.

7. Now copy the software tree of the subset to `/install/mysub/new`, and on the last diskette, you should copy the file called `mysub.name` to `/install/mysub/new/usr/options`.
8. Unmount the diskette by typing:

```
cd / ; umount /install
```

2. BUILDING A SUBSET WITH DRIVERS

This section assumes that you have read and understood the concepts in the rest of this document. To make a 386/ix subset that contains device drivers without using the tools, follow the steps mentioned in §3.7, without unmounting the diskette, and in addition, execute the following steps:

1. Make sure you have separate directories for all the drivers belonging to the subset, each containing at least a `Driver.o`, `Master`, and `System` file.
2. After having copied the software tree, WITH THE EXCEPTION OF THE DRIVER MODULES, copy all driver module directories to `/install/mysub/driver`, *not* to `/install/mysub/new`.

3. Create a file containing the names of each driver directory, one per line, and copy it to `/install/mysub/install/drivers`.
4. Unmount the diskette by typing:

```
cd / ; umount /install
```

Appendix D: RELATED MANUAL ENTRIES

This appendix contains 386/ix Operating System manual entries of interest when using this document.

NAME

idinstall – add, delete, update, or get device driver configuration data

SYNOPSIS

/etc/conf/bin/idinstall –[**adug**] [–**e**] –[**msoptnirhcl**] –**R**dir dev_name

DESCRIPTION

The *idinstall* command is called by a Driver Software Package (DSP) Install script or Remove script to Add (–**a**), Delete (–**d**), Update (–**u**), or Get (–**g**) device driver configuration data. *Idinstall* expects to find driver component files in the current directory. When components are installed or updated, they are moved or appended to files in the **/etc/conf** directory and then deleted from the current directory unless the –**k** flag is used. The options for the command are as follows:

Action Specifiers:

- –**a** Add the DSP components
- –**d** Remove the DSP components
- –**u** Update the DSP components
- –**g** Get the DSP components (print to **std out**, except Master)

Component Specifiers (if no component is specified, the default is all options except for the –**g** option, where a single component must be specified explicitly):

- –**m** Master component
- –**s** System component
- –**o** Driver.o component
- –**p** Space.c component
- –**t** Stubs.c component
- –**n** Node (special file) component
- –**i** Inittab component
- –**r** Device Initialization (rc) component
- –**h** Device shutdown (sd) component
- –**c** Mfsys component: file system type config (Master) data
- –**l** Sfsys component: file system type local (System) data
- –**z** Define component: list of preprocessor symbols needed to compile this module

Miscellaneous:

- –**e** Disable free disk space check
- –**k** Keep files (do not remove from current directory) upon add or update.

In the simplest case of installing a new DSP, the command syntax used by the DSP's Install script should be:

idinstall –a *dev_name*

In this case the command will require and install a Driver.o, Master and System entry, and optionally install the Space.c, Stubs.c, Node, Init, Rc, Shutdown, Mfsys, and Sfsys components if those modules are present in the current directory.

The **Driver.o**, **Space.c**, and **Stubs.c** files are moved to a directory in **/etc/conf/pack.d**. The *dev_name* is passed as an argument, which is used as the directory name. The remaining components are stored in the corresponding directories under **/etc/conf** in a file whose name is **dev_name**. For example, the Node file would be moved to **/etc/conf/node.d/dev_name**.

The *idinstall* **−m** usage provides an interface to the *idmaster* command which will add, delete, and update *mdevice* file entries using a Master file from the local directory. An interface is provided here so that driver writers have a consistent interface to install any DSP component.

As stated above, driver writers will generally use only the *idinstall* **−a** *dev_name* form of the command. Other options of *idinstall* are provided to allow an update DSP (i.e., one that replaces an existing device driver component) to be installed, and to support installation of multiple controller boards of the same type.

If the call to *idinstall* uses the **−u** (update) option, it will:

Overlay the files of the old DSP with the files of the new DSP.

Invoke the *idmaster* command with the "update" option if a Master module is part of the new DSP.

Idinstall also does a verification that enough free disk space is available to start the reconfiguration process. This is done by calling the *idspace* command. *Idinstall* will fail if insufficient space exists, and exit with a non-zero return code. The **−e** option bypasses this check.

Idinstall makes a record of the last device installed in a file (**/etc/.last_dev_add**) and saves all removed files from the last delete operation in a directory (**/etc/.last_dev_del**). These files are recovered by **/etc/conf/bin/idmkenv** whenever it is determined that a system reconfiguration was aborted due to a power failure or unexpected system reboot.

ERROR MESSAGES

An exit value of zero indicates success. If an error was encountered, *idinstall* will exit with a non-zero value and report an error message. All error messages are designed to be self-explanatory. Typical error messages that can be generated by *idinstall* are as follows:

Device package already exists.
Cannot make the driver package directory.
Cannot remove driver package directory.
Local directory does not contain a Driver object (**Driver.o**) file.
Local directory does not contain a Master file.
Local directory does not contain a System file.
Cannot remove driver entry.

SEE ALSO

idspace(1M), idcheck(1M);
mdevice(4), sdevice(4) in the *Programmer's Reference Manual.*

NAME

idmkinit – read files containing specifications, 386/ix addendum

ENHANCED FUNCTIONALITY

Files in **/etc/conf/init.d** that have the same name as a DSP driver will only be included in **/etc/inittab** if the driver is configured. All of the other files in **/etc/conf/init.d** will always be included.

EXAMPLE

/etc/conf/init.d/foo will always be included in **/etc/inittab**, providing there is no driver named **foo**.

/etc/conf/init.d/asy will only be included in **/etc/inittab** if **asy** is configured in the **sdevice** file.

NAME

insdriver – install kernel driver files

SYNOPSIS

insdriver [**–r** root_directory] [**–d** driver_directory]
[**–n** driver_name]

DESCRIPTION

The *insdriver* command creates the kernel module directory, copies the driver files into it, and optionally, executes **/etc/kconfig** to configure and build a kernel.

By default, the configuration directory is **/etc/conf**; this may be overridden by setting the environment variable **$ROOT**, or by using the **–r** option on the command line. Setting **$ROOT** will cause the configuration directory to become **$ROOT/etc/conf**. The **$CONF** environment variable is no longer used by *insdriver*, *inskern*, or *kconfig* (see *inskern*(1) and *kconfig*(1)). This requires that the directory **/etc/conf** exist in $ROOT.

For a description of the configuration directory and kernel module files, see chapter 3 of the *Integrated Software Development Guide*.

The driver directory is the directory currently containing the driver files. If not supplied on the command line, *insdriver* queries for the driver directory.

If the driver name is not supplied on the command line, *insdriver* queries for the name. This name must be the name of the driver object file (**.o** file) and is the name that will be used in the system file modules list.

Insdriver queries for a one-line description of the driver. This description is entered in **$CONF/modules/description** for use by the *kconfig* command.

After the files in the driver directory are copied to **$CONF/modules/***driver_name* and the description file is updated, *insdriver* queries for kernel build and, on verification, executes **/etc/kconfig**.

EXAMPLES

```
insdriver -c /tmp/tstconf -d mousedir -n mouse
```

The driver files in the directory *mousedir* will be copied to **/tmp/tstconf/modules/mouse**.

FILES

$CONF/modules/description

SEE ALSO

inskern(1), kconfig(1), mkunix(1M),
Integrated Software Development Guide.

NAME

kconfig – configure, build, and install a kernel

SYNOPSIS

kconfig [**–r** root_directory]

DESCRIPTION

The *kconfig* command provides a menu interface to configure, build, or install a kernel.

By default, the root of the directory tree in which the configuration takes place is **/etc/conf**; this may be overridden by setting the environment variable **$ROOT** or by using the **–r** option on the command line. This root will be referred to as **$ROOT** throughout this manual entry.

System files are contained in **$ROOT/etc/conf/cf.d**. The *kconfig* command performs all modifications to the system files via menu choices.

The possible responses to *kconfig* menus are:

a number corresponding to the menu item choice
an **m**, to return to the previous menu
a **q**, to quit and exit the program

Each of the above must be followed by RETURN.

Where a *kconfig* query ends with text within parentheses, that text is the default. Using RETURN will select the default. Otherwise, enter the requested information, followed by RETURN.

Menus

If the **$ROOT** directory has not been specified, *kconfig* will query for the needed information before the first menu appears.

Top Level Menu

```
MAIN MENU

 1) CONFIGURE KERNEL
 2) BUILD A KERNEL
 3) INSTALL A KERNEL

Enter Choice [1-3,q]:
```

Choice **1** will cause the Kernel Configuration Menu to be displayed (see Kernel Configuration Menu below).

Choice **2** will build the kernel by executing **$ROOT/etc/conf/bin/idbuild**. The kernel will be built as configured in choice **1**. If the kernel build is successful, *kconfig* will query for kernel installation. If the kernel is to be installed, **/etc/inskern** will be executed (see *inskern*(1)).

Choice **3** will put up the Install Kernel Menu, which is used to install a previously built kernel (see Install Kernel Menu below).

Install Kernel Menu

Before displaying the menu, *kconfig* prints a notice that a system shutdown is required to install a kernel (see *shutdown*(1M)). This menu displays all kernels contained in the **$ROOT/etc/conf/kconfig.d**. directory. For example:

```
CHOOSE THE KERNEL TO INSTALL

 1) unix.1
 2) unix.2

Enter Choice [1-2,m,q]:
```

The number chosen indicates which kernel will be installed. After verifying the choice, *kconfig* queries for the shutdown grace period and executes **/etc/inskern** to install the kernel.

Kernel Configuration Menu

The kernel configuration menu presents menu choices for modifying system file(s). After returning from the Configuration Menu, if modifications were made, *kconfig* asks whether to save the modified system file(s).

```
CONFIGURATION MENU

 1) ADD DRIVER
 2) REMOVE DRIVER
 3) ADD FACILITY
 4) REMOVE FACILITY
 5) ADD DEFAULT PARAMETERS FOR MEMORY SIZE
 6) ADD TUNABLE PARAMETERS
 7) CONFIGURE HIGH PERFORMANCE DISK DRIVER

Enter Choice [1-7,m,q]:
```

Choice **1** puts up the Add Driver Menu, which is used to include kernel device driver modules in the set of configured modules (see Add Driver Menu below).

Choice **2** puts up the Remove Driver Menu, which is used to remove kernel device driver modules from the set of configured modules (see Remove Driver Menu below).

Choice **3** puts up the Add Facility Menu, which is used to include groups of kernel modules in the set of configured modules (see Add Facility Menu below).

Choice **4** puts up the Remove Facility Menu, which is used to remove groups of kernel modules from the set of configured modules (see Remove Facility Menu below).

Choice **5** puts up the Add Default Parameters Menu. This menu is used to set predefined tunable parameters based on system memory size (see Add Default Parameters Menu below).

Choice **6** adds user-entered tunable parameters to the **stune** system file. *Kconfig* queries for the parameter name and value. Using RETURN terminates the additions. If this is a new parameter (one that is not already in the system file **$ROOT/cf.d/mtune**), *kconfig* will query for minimum, maximum, and default values.

Choice **7** presents a series of inquiries and menus used to configure the High Performance Disk Driver (see "Configure High Performance Disk Driver" below).

Add Driver Menu

This menu lists known drivers that are currently able to be configured in the **sdevice** system file. The menu items are examples only and will vary according to the **sdevice** file contents.

```
CHOOSE A DRIVER TO ADD TO THE CURRENT CONFIGURATION

 1) tty line discipline 0
 2) Mouse driver

Enter Choice [1-2,m,q]:
```

After verifying the choice, *kconfig* includes the corresponding kernel driver module in the **sdevice** system file. It does this by setting the second field to **Y** for that entry.

Remove Driver Menu

This menu lists drivers that can be removed from the **sdevice** system file. These drivers are modules currently included in the **sdevice** file (entries having the second field set to **Y**). The menu items are examples only and will vary according to the **sdevice** file contents.

```
CHOOSE A DRIVER TO REMOVE FROM THE CURRENT CONFIGURATION

 1) AT serial I/O Driver
 2) AT floppy disk driver
 3) AT hard disk diskdriver
 4) AT Line Printer Driver
 5) AT Wangtek cartridge tape driver
 6) AT Bell Technologies Hub board
 7) Shell Layers Driver

 Enter Choice [1-7,m,q]:
```

After verifying the choice, *kconfig* removes the corresponding kernel driver module from the system file by setting the second field to **N** for that entry.

Add Facility Menu

This menu lists kernel facilities whose modules are not in the set of currently configured modules. The menu items are examples only and will vary according to the contents of this file.

```
CHOOSE A FACILITY TO ADD TO THE CURRENT CONFIGURATION

 1) Kernel Debugger
 2) Unix Kernel Profiler
 3) SunRiver Fiber Optic Station
 4) 386/ix TCP/IP

Enter Choice [1-2,m,q]:
```

After verifying the choice, *kconfig* adds the corresponding group of kernel modules to the **sdevice** file by setting the second field to **Y** for that entry.

Remove Facility Menu

This menu lists kernel facilities whose modules are in the current **sdevice** file. The menu items are examples only and will vary according to the system file contents.

```
CHOOSE A FACILITY TO REMOVE FROM THE CURRENT CONFIGURATION

 1) MS-DOS File System Service
 2) Inter Process Communication
 3) Shared Memory

Enter Choice [1-3,m,q]:
```

After verifying the choice, *kconfig* removes the corresponding group of kernel modules from the system file.

Add Default Parameters Menu

This menu lists the available predefined tunable parameter sets that are based on memory size. The actual list may differ from the one shown here.

```
CHOOSE THE CLOSEST MEMORY SIZE

 1) 2MB
 2) 4MB
 3) 8MB
 4) 16MB

Enter Choice [1-4,m,q]:
```

The standard kernel supplied with the system is optimized for a system with only 2 MB of RAM . Although all memory that is located when 386/ix boots will be used, performance will increase if memory can be dedicated to system buffers and other kernel structures. This option allows you to tune the kernel to make more efficient use of different amounts of system memory. If the amount of memory you have installed falls between the available choices, choose the next lowest option. The 386/ix Operating System will operate unreliably if the system does not have as much memory as the kernel expects.

Configure High Performance Disk Driver

This section summarizes the inquiries and menus that will be presented. See section 11.3 of "386/ix Maintenance Procedures" for a complete discussion of the configuration restrictions and related information, as well as a more in-depth step-by-step presentation of the configuration process.

Step 1.

The initial series of inquiries and menus concern the fixed disk controller configuration. The following is the decision path for the top-level questions:

```
Is there a standard AT controller? (ST-506, RLL, ESDI) (y):

 If yes:  Do you have a secondary AT controller? (y):
          If no:   Do you have a SCSI host controller? (y):

 If no:   Do you have a SCSI host controller? (y):
```

The following are the menus and inquiries that will be presented if the response is **y** to either question regarding a SCSI host controller:

```
SCSI CONTROLLER SUPPORTED

 1) FUTURE DOMAIN
 2) ADAPTEC

Enter Choice [1-2,q]:
```

If the choice from the previous menu was **2**, the following menu will be displayed (note that the actual menu may not contain all of these choices; refer to the discussion in section 11.3.1 of "386/ix Maintenance Procedures"):

```
INTERRUPT VECTOR CONTROLLER SETTING

 1) INTERRUPT VECTOR  5
 2) INTERRUPT VECTOR  11
 3) INTERRUPT VECTOR  14
 4) INTERRUPT VECTOR  15

Enter Choice [1-4,q]:
```

The following question will be asked (for both types of SCSI controllers):

```
Is there a SCSI tape drive? (y):
```

Step 2.

The next series of questions and menus concern configuration of a RAM disk:

```
Do you want a RAM disk? (y):
```

If the response is **y**, the following menu will be displayed:

```
RAM DISK SIZE (in 4K blocks)

 1 )    256   (1   MB)   10)   1408   (5.5 MB)
 2 )    384   (1.5 MB)   11)   1536   (6   MB)
 3 )    512   (2   MB)   12)   1664   (6.5 MB)
 4 )    640   (2.5 MB)   13)   1792   (7   MB)
 5 )    768   (3   MB)   14)   1920   (7.5 MB)
 6 )    896   (3.5 MB)   15)   2048   (8   MB)
 7 )   1024   (4   MB)   16)   2176   (8.5 MB)
 8 )   1152   (4.5 MB)   17)   other
 9 )   1280   (5   MB)

Enter Choice [1-17,q]:
```

If the choice from the previous menu was **17**, the following will be displayed:

```
Specify the size in 4K blocks (do NOT specify in MB):
```

Step 3.

The selected configuration will be presented for verification. The configuration shown below is an example only.

```
The configuration selected is as follows:

Primary:
    Standard AT controller
    Number of disk drives:          2   (maximum supported)
    Interrupt vector:              14

Secondary:
    SCSI controller, Future Domain
    Number of disk drives:          4   (maximum supported)
    Interrupt vector:               5
    SCSI tape drive

RAM disk:  256 4K blocks
```

The following question will then be asked:

```
IS THIS THE DESIRED CONFIGURATION (y):
```

If the response is **y**, the following will be displayed:

```
High Performance Disk Driver configured.
```

If the response is **n**, there is an opportunity to configure it again:

```
DO YOU WANT TO START OVER (y):
```

will be displayed.

Description File

The kernel module descriptions and group designations used by the configuration menus are obtained from the file **$ROOT/etc/conf/kconfig.d/description**. This file lists the kernel module names and description and the group designations.

Description file format

module name	–	–	group name	–	description

Example

asy	–	–	io	–	AT serial I/O Driver
hub	–	–	io	–	AT Bell Technologies Hub board
lp	–	–	io	–	AT Line Printer Driver
sxt	–	–	io	–	Shell Layers Driver
wt	–	–	io	–	AT Wangtek cartridge tape driver
–	–	–	ipc	–	Inter Process Communication
ipc	–	–	ipc	–	ipc common routines
msg	–	–	ipc	–	ipc message facility
sem	–	–	ipc	–	ipc semaphore facility

The module name is the name of the system file entry. The group name is the group for that module; group name *io* designates I/O drivers. A "–" in the module name designates the description used on the menus for the designated group.

For example, the group name *ipc* designates a kernel facility. The description field on the line with module name "–" and group name *ipc* is the description used on the Add or Remove Facilities Menus. The lines with module names and group *ipc* designate the module list for that group. In this case, adding or removing the facility *Inter Process Communication* would add or remove the modules *ipc*, *msg*, and *sem*.

Note that the "–" columns in the above example must be present. These fields are reserved for future function information.

FILES

$ROOT/etc/conf/kconfig.d

SEE ALSO

inskern(1), shutdown(1M),
"386/ix Maintenance Procedures."

WARNINGS

The *kconfig* command can be executed only by user **root**. To install a kernel, *kconfig* should be executed from the / (root) directory.


INTERACTIVE
A Kodak Company