



INTERIX™ for Microsoft® WINDOWSNT®

INTERIX Release 2.2 Commands and Utilities User's Guide

Softway Systems, Inc.

185 Berry Street, Suite 5514

San Francisco, CA 94107

Tel. (415) 896-0708

Fax (415) 896-0709

<http://www.interix.com>



INTERIX Commands and Utilities User's Guide, Version 2.2

Written by John McMullen.

Copyright 1996, 1997, 1998 Softway Systems, Inc. All rights reserved. You may not copy this document in whole or in part without prior written permission from Softway Systems, Inc.

Copyright © 1989, 1993 The Regents of the University of California. All rights reserved. Portions of this distribution are derived from code owned by the Regents of the University of California.

Portions of the software distribution are derived from Free Software Foundation, Inc. source code. As required by the FSF license, changes to the source code are available from the INTERIX web site's tool warehouse. The URL is <http://www.interix.com/tw/>.

Softway Systems makes no representations or warranties other than the standard warranty with regard to the contents of this manual or to the suitability of this software product for any particular purpose.

Softway Systems reserves the right to make any changes to this software product and to revise the information about the products contained in this guide without an obligation to notify any persons about such revisions or changes.

Trademarks

INTERIX is a trademark of Softway Systems, Inc.

Microsoft and Windows NT are either registered trademarks or trademarks of Microsoft Corporation in the United States and/or other countries.

Adobe and Acrobat are trademarks of Adobe Systems Incorporated which may be registered in certain jurisdictions.

Pentium is a registered trademark of Intel Corporation.

UNIX is a registered trademark in the United States and other countries, licensed exclusively through The Open Group.

Open Software Foundation, OSF, OSF/Motif, and Motif are trademarks of The Open Group.

X Window System is a trademark of X Consortium, Inc.

All other products mentioned are trademarks or registered trademarks of their respective companies.

Contents

Introduction	7
Welcome	7
What INTERIX Is	7
OPENNT to INTERIX	8
Installation and Upgrading	9
Contacting Technical Support	9
Getting Started	11
Getting Up and Running	11
Setting Up Your Workspace	12
Selecting a Shell	12
Emacs Command-Line Editing in Shells	13
Environment Variables	13
Customizing Your Startup Files	14
Installing Desktop Shortcuts	15
Other Customizations	17
The INTERIX Environment	19
Overview	19
Features of INTERIX	20
Simultaneous Users	20
Shells	21
Job Control.....	22
Device Support: /dev/null, /dev/tty, and /dev/tape	22
Locale Support	23
INTERIX Specific Behavior	23
Pathnames On INTERIX	24
The Super User and Appropriate Privileges	24
Security on Windows NT	25
Domains and User Names	25
File Permissions	26
Windows NT Object Security	27
Traverse Checking of Directory Permissions	29
The File System	30
Converting Between POSIX and MS-DOS Text Format	31
Working with Win32 Programs	33
Invoking Win32 Programs	33
Adding Directories to Your PATH	35
Aliases	35
The Generic Windows NT System Directory	37

Running CMD.EXE Built-In Commands	37
Links	38
Shell Scripts	38
Stopping Win32 Programs	39
File Redirection	40
Avoiding INTERIX / Win32 Conflicts	40
Terminals and Other Devices	43
The INTERIX Terminal	43
Enabling Console Properties	44
Enabling Windows NT Cut and Paste	45
Setting Screen Colors	45
Enabling Scroll Bars	47
Changing the Terminal Title Bar	48
Alternate Terminal Window Sizes	49
Virtual Devices	51
Tape Drives	52
Tape Portability	52
Services and Daemons	55
Overview	56
The Cron Service	56
Setting Up Cron	57
Scheduling Jobs Using Crontab	57
Scheduling Jobs Using At and Batch	58
Administering Other Services	59
The Telnet Daemon	60
Installing the Telnet Daemon	60
Using Telnet and Telnetd	61
The Syslog Daemon	61
The syslog.conf File	62
Other Documentation	65
The On-Line Library	65
The Reference Pages	66
The Man Command	67
Searching for Keywords	68
The Acrobat Reader	68
Viewing on a 640x480 Display	68
Technical Notes	69
X Window Clients	71
Setting Up X11 Applications	71
Running X11 Applications	72
The MWM Window Manager	73
Further Reading	73

POSIX Regular Expressions	75
Introduction to Regular Expressions	75
Basic Regular Expressions	76
Extended Regular Expressions	78
Character Sets	78
Examples	80
Basic Regular Expressions	80
Extended Regular Expressions	81
Glossary	83
Index	91

WELCOME

WHAT INTERIX IS

HARDWARE/SOFTWARE REQUIREMENTS

CONTACTING TECHNICAL SUPPORT

CHAPTER 1

Introduction

Welcome

Thank you for purchasing INTERIX. The INTERIX subsystem, commands and utilities provide the most-requested features and functionality of a UNIX system environment for your Windows NT machine. The INTERIX subsystem and the commands and utilities environment is provided with INTERIX Workstation Lite, INTERIX Workstation, and INTERIX Server.

This User Guide introduces that environment. Moving to the INTERIX environment is no more difficult than switching between different UNIX systems. For complete details on all commands and utilities features and functions, see the reference pages.

What INTERIX Is

INTERIX is a complete subsystem environment for your Windows NT machine. It conforms to the POSIX.1 and POSIX.2 standards; these standards describe operating system service interfaces and utilities such as the shell, *awk*, *vi*, and so on.

INTERIX is about portability—it’s about a consistent set of interfaces that let users and applications move between platforms. This portability is what is meant by “open systems.” Portable, open systems are built of different industry standards, such as the ISO / IEEE POSIX standards and the ISO / ANSI C-language standard. An industry specification, such as The Open Group’s *Single UNIX Specification*, incorporates those standards and adds other feature definitions. INTERIX implements the most-requested features of these standards and specifications on Windows NT, closely integrated with all the functionality present in Microsoft’s newest operating system environment.

With INTERIX, you have the important features of popular open systems platforms, including pipes, hard file links, networking services, and X11 graphical support through the X Window System. You get utilities such as KornShell, *awk*, and *vi*. Shell scripts and other scripted applications will run under INTERIX.

The INTERIX environment is not an emulation; it is a separate environment subsystem that integrates with the Windows NT kernel, just as the Win32 subsystem does. When you install the INTERIX Workstation Lite, INTERIX Workstation, or INTERIX Server, you install a new extended subsystem that replaces the POSIX subsystem provided by Microsoft with Windows NT. The INTERIX extended subsystem provides all of the features described in the POSIX.1 standard, and more besides.

Chapter 3, “The INTERIX Environment,” describes the architecture of INTERIX and Windows NT subsystems in more detail.

OPENNT to INTERIX

Earlier releases of INTERIX, before release 2.2, went under the name OPENNT. To preserve backwards compatibility throughout the 2.x releases, the OPENNT pathnames, environment variables, and compiler macros stay the same. The name of the installation directory is still available through the environment variable *OPENNT_ROOT*.

Eventually, future releases will migrate to new names (such as renaming *OPENNT_ROOT* to be *INTERIX_ROOT*), but the old ones will remain for backwards compatibility during the 2.x series of releases. For example, if you are installing the product over a previous release, the old installation directory is used. If this is a fresh installation, the directory **C:\Interix** is used. Eventually, occurrences of “OpenNT” will be replaced by “Interix”.

Installation and Upgrading

The INTERIX Release 2.2 *Getting Started* booklet describes the hardware requirements for the INTERIX products, and how to install them.

Take note of the following if you already have an earlier release of INTERIX on your system:

- Before installing the new release into the same root directory as the previous release (normally **//C/Interix**), you should save any text files you've customized (typically **//C/Interix/etc/profile.lcl** or **//C/Interix/share/magic**), copy the customized files to a safe location. Then uninstall the previous release.
- If you install the new release into a different root directory, remember that any desktop shortcuts or icons will still point to the *old* installation. You must manually edit the properties of the shortcuts or icons to point to the new installation.

Contacting Technical Support

Softway Systems, Inc. provides technical support between the hours of 9 a.m. and 5 p.m. Pacific Standard Time. E-mail is the best method for contacting technical support.

Please include your name, a phone number where you can be reached, and a return e-mail address if it is not the same as that from which you sent the e-mail. Your problem description should include the INTERIX version number, the installed components, and your registered serial number.

You can reach Softway Systems by any of the following:

E-mail: support@interix.com

Phone: (519) 571-7104

Fax: (415) 896-0709

FTP server: ftp.interix.com (use anonymous login)

Web Site: <http://www.interix.com/>

Mail: Softway Systems, Inc.
Technical Support
185 Berry Street, Suite 5514
San Francisco, CA 94107
USA

GETTING UP AND RUNNING

SETTING UP YOUR WORKSPACE

CUSTOMIZING YOUR STARTUP FILES

INSTALLING DESKTOP SHORTCUTS

OTHER CUSTOMIZATIONS

CHAPTER 2

Getting Started

This chapter describes the basics for getting started with the commands and utilities.

The examples assume you have installed INTERIX in the default location, **C:\Interix**. Pathnames are written in the POSIX subsystem format: drives are prefixed with a double slash and slash is used as the separator, so **C:\Interix** is referred to as **//C/Interix**.

The install directory for the INTERIX are also stored in the environment variable *OPENNT_ROOT*. In some documentation, you may see pathnames such as **\$OPENNT_ROOT/etc**; this refers to the directory **etc** in the INTERIX install directory.

Getting Up and Running

To start the INTERIX shell, either double-click on the shell icon in the INTERIX Program Group, or from a *CMD.EXE* DOS prompt, type:

```
C:\Interix\bin\ksh -l
```

(That's the letter ell, not the number one.)

This gives you the POSIX-conformant KornShell.

To run the C shell, either choose the C shell icon in the INTERIX program group, or from a *CMD.EXE* DOS prompt, type:

```
C:\Interix\bin\tcsh -l
```

Unless otherwise stated, in this document “the shell” refers to the KornShell.

Setting Up Your Workspace

Your choice of shell for interactive sessions is the most important factor in setting up your workspace, since your shell is your work environment. Inside that shell, you can define your work environment by setting environment variables and defining aliases and functions. This section outlines the choices available and describes the important environment variables for INTERIX.

Selecting a Shell

The commands and utilities contains two shells, *ksh* and *tcsh*. The KornShell, *ksh*, is a POSIX.2-conforming shell, much like the standard Bourne shell. (The binary *sh* is a copy of *ksh*.) The C shell, *tcsh*, is the Tenex C shell. (The binary *csh* is a copy of *tcsh*.) These shells are also shipped as files with a **.exe** extension, so they can be run from Win32 shortcuts.

You select a shell by starting it from the Windows NT desktop, either by running it from the Program Manager or the Task Bar, or by clicking on the shortcut or icon, if you have set one up.

If portability is a concern, do not use the C shell. However, many users (especially those who first learned on BSD or BSD-derived systems) prefer to use the C shell as their interactive shell.

Because the INTERIX shells are usually launched from the Desktop, your “default shell” will usually be the one with a shortcut or icon on the Desktop.

When you login to a system using *telnet*, your login shell depends upon how the Telnet Server on that system is configured. For example, most Win32 telnet servers provide *CMD.EXE* as the default login shell. The INTERIX *telnetd* telnet server provides *ksh* as the login shell. See chapter 6, “Daemons and Services,” for more information about *telnetd*.

Emacs Command-Line Editing in Shells

Both of the shells provide the ability to list, edit, and run previous commands. By default, both use the *vi* commands to display and edit commands (**Esc** enters command-line editing mode, **k** displays the previous command, **\$** moves to the end of the line, and so on). Some users prefer to use the key-commands of the *emacs* editor.

To set *emacs*-style command-line editing in the KornShell, use the command

```
set -o emacs
```

To set *emacs*-style command-line editing in the C Shell, use the command

```
bindkey -e
```

See the on-line reference pages for the shells for more information.

Environment Variables

INTERIX stores information about your system in your environment. Some of the environment variables are inherited from the *Environment* list under *System Properties* of the Control Panel; others are set in a startup file when you login.

The following environment variables are set for you:

<i>HOME</i>	Your home directory; taken from the User Manager.
<i>LOGNAME</i>	Your user name including the domain; taken from the User Manager.
<i>OPENNT_ROOT</i> <i>PATH</i>	The directory in which you installed INTERIX. Directories searched for commands (programs and scripts); taken from the startup file \$OPENNT_ROOT/etc/profile (for the KornShell) or \$OPENNT_ROOT/etc/csh.login (for the C shell).
<i>TERM</i>	Your terminal type; the INTERIX terminal window is type <i>opennt</i> ; see chapter 5, "Terminals and Other Devices".
<i>TMPDIR</i>	A directory for temporary files; taken from the startup file \$OPENNT_ROOT/etc/profile (for the KornShell) or \$OPENNT_ROOT/etc/csh.login (for the C Shell).
<i>USERNAME</i>	Your user name without a domain.

To refer to the value of the environment variable, prefix it by `$` or surround the name of the variable by `${}`—both **`$HOME`** and **`${HOME}`** refer to your home directory.

There are other environment variables; you can see a complete list by using the *env* command.

Customizing Your Startup Files

Some commands, like the shell, search for a startup file when they start and run the commands in that file.

The shell runs the commands in the files **`$OPENNT_ROOT/etc/profile`** and **`$OPENNT_ROOT/etc/profile.lcl`** and then runs the commands in the file **`.profile`** in your home directory, if you have a **`.profile`** file.

- A home directory other than Windows NT's default of **`//C/users/default`** can be defined in the User Manager, in either the **User Properties** dialog or in the **Home Directory** box of the **User Environment Profile** dialog.

Changes that affect all INTERIX users on the system should be made to the file **`$OPENNT_ROOT/etc/profile.lcl`** (because future installations of INTERIX may overwrite **`$OPENNT_ROOT/etc/profile`**). Changes for only one user should be made in that user's **`.profile`** file.

You can create a **`.profile`** file for a user by copying the file **`$OPENNT_ROOT/etc/profile.usr`** to the user's home directory and renaming it to **`.profile`**. The user can then edit the file.

The most common customization is to add directories to your *PATH* variable. For example, you may want to keep personal INTERIX programs and shell scripts in a **`bin`** subdirectory of your home directory. The default *PATH* is (line break inserted for readability):

```
//C/Interix/bin://C/Interix/usr/contrib:
//C/Interix/X11R5/bin
```

Other utilities make use of startup files; they are normally stored in your home directory and have names beginning with `"."` such as **`.nexrc`** (*vi* editor startup file).

Utility	Startup File
<i>csh/tcsh</i>	<code>\$OPENNT_ROOT/etc/csh.cshrc</code> , <code>\$OPENNT_ROOT/etc/csh.login</code> , <code>\$HOME/.cshrc</code> ,

	\$HOME/.login,\$OPENNT_ROOT/etc/csh.logout,
	\$HOME/.logout
<i>ex/vi</i>	\$HOME/.nexrc, \$HOME/.exrc
<i>ksh/sh</i>	\$OPENNT_ROOT/etc/profile,
	\$OPENNT_ROOT/etc/profile.lcl, \$HOME/.profile
<i>mailx</i>	\$OPENNT_ROOT/etc/mailx.rc, \$HOME/.mailrc

For information on a utility's startup files, see the reference page for that utility.

Most INTERIX programs (such as *csh*) require their startup files to be POSIX-style text files—lines should end with a line-feed character, not a carriage-return/line-feed pair. To convert a text file to the INTERIX text format, run the command:

```
flip -u file(s)
```

The programs *ksh*, *make* and *awk* (for the **-f** option only) can read script files in either the INTERIX or MS-DOS format.

Installing Desktop Shortcuts

You can start INTERIX utilities from desktop icons (in Windows NT 3.51) and shortcuts (in Windows NT 4.0).

Create the icon or shortcut as you would normally. You need to edit the command line; in Windows NT 4.0, display the Properties for the shortcut and select the “Shortcut” tab. The command line is in the Target: field.

The command line you use depends upon whether the command you want to run is an INTERIX binary, a shell script, or an INTERIX X client.

If the command you want to run is an INTERIX utility, edit the command line to be:

```
C:\WINNT40\system32\posix.exe /c C:\Interix\bin\cmd options
```

Change the pathnames if your Win32 **system32** directory is located elsewhere, or if you installed the commands and utilities somewhere other than **//C/Interix**.

The *cmd* is the actual command name and command line to be run. Everything after the command name is passed to the command being run, so the arguments to the command should be in the POSIX format (options indicated by - and pathnames using the POSIX format).

If the window opens briefly and displays the message “Error opening terminal” then the program is looking for the *TERM* environment variable. Set it in the Win32 environment using the **System** applet of the Control Panel.

If the command you want to run is an X client, edit the command line to be:

```
psxrun cmd options -display localhost:0.0
```

The *psxrun* program is a Win32 program installed with the commands and utilities. The *cmd* is the absolute pathname of the X client in the Win32 format (for example, “C:\Interix\usr\X11R5\bin\puzzle”). You must include the **-display** option, or the X client will not be able to open the display.

If the command you want to run is a shell script, edit the command line to be:

```
C:\Interix\bin\sh.exe -c script
```

The *script* is the absolute pathname of the shell script, in POSIX format. The *script* must also be executable. To make the script executable, give this command in an INTERIX shell window:

```
chmod a+x script
```

Again, *script* is the pathname of the shell script.

Unless the script is interactive, it will appear and disappear too quickly to read. If you want to read the script’s output, you can add a “wait for input” block to the end of the script, such as:

```
while read input
do
    if [ "$input" = "exit" ]
    then
        exit
    fi
echo Type "exit" to leave.
done
```

Other Customizations

INTERIX is a flexible environment which permits a great deal of customization. Other sections of this *User's Guide* discuss some of them, particularly chapter 4, "Working with Win32 Programs," and chapter 5, "Terminals and Other Devices."

- Running Win32 programs—see chapter 4, "Working with Win32 Programs."
- Changing the terminal's window size—see "Alternate Terminal Sizes" in chapter 5, "Terminals and Other Devices."
- Displaying the current directory in the terminal title bar—see "Changing the Terminal Title Bar" in chapter 5, "Terminals and Other Devices."
- Setting terminal display colors—see "Setting Screen Colors" in chapter 5, "Terminals."
- Enabling Windows NT cut-and-paste—see "Enabling Windows NT Cut and Paste" in chapter 5, "Terminals and Other Devices."
- Enabling scroll bars—see "Enabling Scroll Bars" in chapter 5, "Terminals and Other Devices."
- Running the shell in POSIX-conformance mode—see "POSIX Mode" in chapter 3, "The INTERIX Environment."

You can also customize commands with shell scripts, shell functions, and aliases. In the KornShell:

- A **shell script** is a series of commands stored in a file, to be run by the shell. It is a program written in the shell language.
- A **shell function** is also a series of commands to be run by the shell. It is generally less complicated than a shell script and is usually smaller, since it's stored in the current shell rather than being run from a separate file. There are no shell functions in the C shell; C shell aliases do the same thing.
- An **alias** is an abbreviation for a command.

See the on-line reference pages describing the shells for more information.

OVERVIEW

FEATURES OF INTERIX

INTERIX SPECIFIC BEHAVIOR

SECURITY ON WINDOWS NT

THE FILE SYSTEM

CHAPTER 3

The INTERIX Environment

This chapter describes the INTERIX environment, including the underlying Windows NT architecture, some of the features of the INTERIX environment, and a discussion of INTERIX-SPECIFIC behaviors. The architectural information is essential to understanding the interactions between the INTERIX subsystem and the Win32 subsystem (described in chapter 4, “Running Win32 Programs”).

Overview

The INTERIX environment is built upon the INTERIX subsystem, a replacement for the Windows NT POSIX subsystem. The INTERIX subsystem provides many additional features and functionality.

Windows NT is built to provide more than one set of programming interfaces to the operating system kernel. Each of these sets of interfaces is called an “environment subsystem.” The familiar Win32 environment is one subsystem; the POSIX and INTERIX environments are another. Both the INTERIX and Win32 subsystems can run at the same time.

Other subsystems are provided with Windows NT for other purposes, such as the security subsystem; these subsystems are “integral sub-

systems.” The security subsystem and the I/O subsystem are described in this chapter.

The INTERIX and Win32 environments provide roughly the same speed of performance. Some INTERIX applications run faster than the equivalent Win32 application, and some Win32 applications run faster than the equivalent INTERIX application.

One of the networking features of Windows NT is the use of “domains” (which are different from internetwork domains). A domain is a set of Windows NT machines which use a single password database. The existence of Windows NT domains has effects even in the INTERIX subsystem; see the discussion of Windows NT security for more information.

Features of INTERIX

The INTERIX subsystem provides a POSIX system environment for your Windows NT machine. That includes all features dictated by the POSIX.1 and POSIX.2 standards, derived from the 4.2BSD, 4.3BSD, Version 6, and System III operating systems.

In particular, INTERIX features:

- Simultaneous users
- Shells such as the KornShell and the C Shell
- Complete job control
- Device support (**/dev/null**, **/dev/tty**, **/dev/tape**, **/dev/pty**)
- Sockets support
- Shared memory, message queues, semaphores

Simultaneous Users

Multiple users may login to a Windows NT machine (the host machine) by using the *telnet* command from another machine on the network. A telnet server must be installed and running on the host machine, and the user must already have an account on the host machine.

The *telnet* user’s shell depends upon the telnet server installed on the host. If the telnet server is the INTERIX *telnetd*, then the user’s shell will be the KornShell. With other telnet servers, it may be possible to use

any INTERIX shell as the login shell. For more details, see the documentation for your telnet server.

Shells

Users may use either the KornShell (*ksh*) or the C shell (*tcsh*), as described in chapter 2, “Getting Started.” The shells support all of the usual features, including job control, command pipelines, and *#!* script interpretation.

Many people prefer the C shell as their interactive shell, but use the Bourne shell or the KornShell as their script processor. The KornShell is a superset of the traditional Bourne shell, and can conform to the requirements for a POSIX.2 shell.

POSIX Mode Behavior

By default, the KornShell follows traditional KornShell behavior, which is nearly but not quite identical to POSIX behavior. Some work environments require strict POSIX conformance; the differences are listed on the *ksh* reference page under the heading “POSIX Mode”.

To get full POSIX.2 conformance from the KornShell, you must run it in “POSIX Mode”. To run the KornShell in POSIX mode, either invoke the KornShell with the option **-o posix**, run the command **set -o posix** from within the shell, or set the shell parameter *POSIXLY_CORRECT*.

Extensions to the POSIX Shell

The KornShell also provides extensions to the POSIX shell. Below is a list of features you should avoid for a strictly-portable POSIX.2 shell application:

- The KornShell allows you to define a shell function with the syntax

```
function foo {  
    .... body of function  
}
```

- The KornShell supports the **select** control structure.
- The KornShell supports code blocks (**{...}**).

- The KornShell grammar supports a **[[...]]** syntax for string and file conditionals. The portable way to test these is with the *test* utility.
- The KornShell supports a **((...))** syntax for arithmetic conditionals. The portable way to test these is with the *test* utility.
- The KornShell adds the commands *typeset* and *let*.

The Restricted Shell

If the KornShell is invoked under the name *rsh* (either through a hard link or a copy of the binary), it behaves as a *restricted shell*. In a restricted shell:

- The *cd* command is disabled.
- The *SHELL*, *ENV*, and *PATH* environment variables can't be changed.
- Command pathnames can't contain absolute or relative paths.
- The **-p** option of the *command* built-in can't be used.
- File redirections that create files (such as *>*, *>>*, *<>* or *>|*) can't be used.

By default, no shell binary named *rsh* is installed. The *rsh* name for the restricted shell conflicts with the *rsh* remote shell utility. If you want to run a restricted shell, it's better to invoke the shell with the **-r** option.

Job Control

The INTERIX shells support job control: you can stop a job with Control-Z and then restart it in the foreground (using the *fg* command), the background (using the *bg* command), or kill it (using the *kill* command). You can display the current list of jobs with the *jobs* command.

Device Support: */dev/null*, */dev/tty*, and */dev/tape*

Unlike the Win32 environment, the null device is named ***/dev/null***. (Under Win32, the null device is named NUL.) You can redirect input and output from and to ***/dev/null***.

Terminal devices also exist and can be listed with the *tty* command. The controlling terminal ***/dev/tty*** exists, as do terminals named

/dev/ttynxx, where *x* is a digit. See chapter 5, “Terminals and Other Devices,” for more about the INTERIX terminal window. Pseudo-terminal devices are supported, so utilities such as *xterm* run on INTERIX.

INTERIX also supports tape devices. The tape devices are accessed through the file names **/dev/tapex** (or **/dev/ntapex**, if you want to open the tape drive in non-rewind mode), where *x* is a digit from 0-9. You must install the tape device through the Windows NT **Tape Devices** applet in the Control Panel before you can access it through the INTERIX environment

If you have trouble reading tape archives written on another system, see the INTERIX *mt* reference page for suggestions.

Locale Support

Currently, INTERIX supports only the C and POSIX locales, as defined in ISO/ANSI C. Support for additional locales and for custom locale definitions is planned for a future release.

European keyboards are supported. Because the locales are not yet supported, not all utilities will recognize the extended characters. Only the character set for the POSIX locale is guaranteed to work. That is, you can create a file containing (for example) ö in its name, but the *ls* utility will display ? instead of ö, because ö is not a character in the POSIX locale.

INTERIX Specific Behavior

The INTERIX environment conforms to the POSIX standards (1003.1 and 1003.2). Three behaviors of the INTERIX environment may surprise users of open systems:

- Pathnames can contain drive letters
- INTERIX has no super user
- User authentication is different

User authentication is discussed in the section “Security on Windows NT.”

Pathnames On INTERIX

Pathnames on INTERIX and POSIX use / as the pathname separator. Absolute pathnames on INTERIX contain drive letters. The drive letter is indicated at the beginning of the pathname by // and must be uppercase: the Win32 pathname **C:\Interix** becomes **//C/Interix** on INTERIX.

A pathname that begins with / (one slash) is taken to be at the root of the current drive. A pathname beginning with more than two slashes is treated as if it begins with one slash.

The character set for filenames on INTERIX is the set specified by POSIX: All printable characters are allowed in filenames except for

/ \ : * ? " | '

The directory name **\$psxjunk** is reserved for use by the subsystem. It's created to store files which are removed while being used by a process.

INTERIX doesn't yet follow the Universal Naming Convention (UNC).

The Super User and Appropriate Privileges

On many systems, there is a super user, the “root” user, who can do anything. There is no such user in POSIX or on INTERIX. When the POSIX standard was being set, the concept of a super user was felt to be an administration issue. Instead, the POSIX standard defines the concept of “appropriate privileges” for some functions.

Appropriate privileges are typically required for actions such as file system access, for certain process control actions (sending signals to other processes), or for changing the effective user ID or group ID of a process in such a way that it changes the ability of the process to perform certain actions.

According to POSIX, a file's permissions include a set-user ID bit and a set-group-id bit. If either or both of those bits are set on a file and a process executes that file, the process gains the user or group ID of the file. There are no appropriate privileges on Windows NT to set the setuid or setgid mode bits on a file.

Security on Windows NT

The Windows NT security model may hold some surprises. For one thing, users and groups share the same namespace in the Security Access database, rather than groups and users being separate.

Within the INTERIX subsystem, file ownership on the New Technology File system (NTFS) behaves as the POSIX standard requires. If you're not going to exchange files between the INTERIX environment and the Win32 environment, you probably won't notice a difference. Understanding Windows NT security model makes it easier to understand interactions between the utilities from one environment and files from another. For example, *chown*, *chgrp*, and *chmod* may not behave as you expect on files created using Win32 applications.

User and group information is stored in the Security Access database. Both users and groups are stored in the same database, but group and user names must be unique; no group can have a user's name and vice versa. (This database replaces */etc/passwd* and */etc/groups* files.) Users and groups are created with the User manager application, found in the Administrative Tools program group, or with the Win32 *net user* command. (Example shell scripts to create and remove users are included in the directory *\$OPENNT_ROOT/usr/examples/admin*.) Users can belong to many groups.

Domains and User Names

All machines served by a single Security Access database form a domain. (Every Windows NT machine also has its own domain for local administration.) Different domains can have users with the same login name. The combination *domainname\loginname* is unique on a network: the user represented by *DEV\chris* is not the user represented by *ADMIN\chris*.

The Windows NT idiom for specifying a domain and a username is *domain\username*. INTERIX utilities return *domain+username*, but will accept any of these forms where a username should go:

domain\username
domain/username
domain:username
domain+username

The form *domain+username* interferes least with other utilities; you don't have to escape any characters and the + character is not used in any INTERIX or Win32 pathnames.

If you specify a *username*, INTERIX treats it as the user with that name in the current logged-in domain.

Before examining the Windows NT security system further, a review the POSIX file permissions scheme is necessary.

File Permissions

In the INTERIX environment, each file has an owner and a group, and three sets of permissions: one set for the owner, one set for members of the file's group, and a last set for everyone else. A set of permissions is read, write, and execute permissions.

You can display the permissions with the long file listing of *ls*:

```
drwxr-xr-x user1 PosixUsers ... /user1
-rwxr-xr-x user1 PosixUsers ... /user1/report.txt
```

In the permissions list (drwxr-xr-x or -rwxr-x--), each character represents a permission bit:

- The first bit indicates the file's type (d for directory, - for regular files, and so on; see the *ls* reference page)
- The next three bits are the read (r), write (w), and execute (x) bits for the file's owner, showing the owner's ability to read, write and execute the file. (For a directory, "execute" means search the directory.) A - indicates the permission is not granted.
- The next three bits are the read (r), write (w) and execute (x) permissions for the file's group. In the example, members of the group other than the owner cannot write to the file *report.txt* but they can read and execute the file.
- The last three bits are the read (r), write (w) and execute (x) permissions for the rest of the world.

In the INTERIX subsystem, the owner is the user who created the file, and the file inherits its group from the group of the directory.

Group names can contain spaces; for example, "Domain Users". While this allows a broader range of names, it will confuse shell and *awk* scripts that expect to parse the group name as a single token.

Windows NT Object Security

All objects in Windows NT on NTFS have an owner and a primary group. (Objects can actually be owned by a group rather than a user; more on this later.) Furthermore, each secure object has an access control list (ACL) made up of access control entries (ACE). Each ACE applies to a particular group or user, and either allows or denies a type of access to that group or user. You can see the ACL for a secure object by using the Win32 program *CACLS.EXE* or by selecting the object and then selecting the Permissions item from the Security menu.

The access types for files are:

Permission	Abbreviation
Read	R
Write	W
Execute	X
Delete	D
Change Permissions	P
Take Ownership	O

(There are actually many more permissions; the *CACLS.EXE* program lists all of the permissions associated with a file.)

Each user or group with access has its permissions listed with respect to the object. The most commonly-used combinations have been given names:

Name	Permissions
Read	RX
Change	RWXD
Full Control	All—RWXDOP
No Access	Explicit denial for the group or user
Special Access	Combination of permissions without a name

Directories have two groups of permissions, one for the directory itself and the other for the files within the directory:

Name	Directory	File Access Types
List	RX	Not specified
Read	RX	RX
Add	WX	Not specified
Add & Read	RWX	RX
Change	RWXD	RWXD
Full Control	All	All
No Access	Explicit denial for the group or user	

Special Directory Access Combination without a name
Special File Access Combination without a name

A file created through the INTERIX subsystem and viewed with **ls -l** has the following permissions and attributes:

- The file is owned by the user who created it.
- The file's group is the same as the group of the directory.
- File permissions are dictated by the file creation mask and the user mask (the "umask"; see the reference page for the *umask* utility).

POSIX files are given three ACEs, one for the owner, one for the group, and one for the group "Everyone", which represents everyone else. POSIX permissions are represented as follows:

- The POSIX read permission is represented by the Windows NT permission read (R).
- The POSIX write permission is represented by the Windows NT permission write (W). If the file's Read Only attribute is set, INTERIX doesn't assign write permission, regardless of the contents of the ACEs. (Using *chmod* to assign write permission to a file with the Read Only attribute set unsets the Read Only attribute.)
- The POSIX execute permission is represented by the Windows NT permission execute (X).
- The owner also has permission to change permissions (P) and to take ownership of the file (O).

A file created through the Win32 subsystem may have a different number of ACEs associated with it, and those ACEs may not fit so neatly into the categories of user, group, and other. The INTERIX tools will "assemble" permissions from the available ACEs:

- The file is owned by the user — unless the user is a member of the group "Administrators" in which case the file is owned by the group Administrators.
- The file's group will probably be "Domain users" on a server or "None" on a workstation.
- If the owner has no specific ACE associated with it, then the owner permission bits are empty.
- If the owner is a group, those permissions will be used as the owner's permission, and the group permissions will be empty.

If the ACE used to determine the owner's permissions doesn't have change permission (P) or take ownership permission (O), the utilities *chown*, *chgrp*, and *chmod* may be hampered.

All objects on a FAT or HPFS filesystem are owned by the user "Everyone" and the group "Everyone".

Traverse Checking of Directory Permissions

By default, Windows NT users have the right to change directories even if they do not have permission for all the directories in the path. On POSIX systems, a user cannot change into a directory unless he or she has permission to enter all of the directories in the path. The POSIX behavior is termed "traverse checking" in Windows NT documentation.

For example, if you don't have permission to enter a directory, you cannot enter any of its children, even if you do have permission on those directories. The default Windows NT behavior allows you to enter any directory where you have permission. The POSIX standard requires you to have permission on all directories in the path.

To ensure the POSIX behavior, remove the "Bypass Traverse Checking" permission for all users:

1. Login as Administrator.
2. Open the User Manager.
3. From the Policies menu, select User Rights
4. Ensure the Show Advanced User Rights button is selected.
5. Select the user "Everyone".
6. Select the Bypass Traverse Checking right.
7. Select the **Remove** button.
8. Select the **OK** button.

Traverse checking is only supported on NTFS filesystems. Removing the traverse checking right has no effect on FAT filesystems.

The File System

For the behavior most like that described in the *Single UNIX Specification*, you should install and use INTERIX on a New Technology File System (NTFS). NTFS supports all of the requirements of a POSIX file system.

- Pathnames use / as the pathname separator.
- Pathnames starting with //D, where D is an uppercase letter, refer to files on a particular drive. For example, pathnames beginning with //C are on the C: drive.
- Filenames are case-sensitive (**Temp**, **temp**, and **TEMP** are all different files and the system can distinguish between them).

On a FAT filesystem or across a network, file names are case-insensitive, though case is generally preserved. For reasons internal to Windows NT, file specifications using shell wildcards (such as ? or *) are case-sensitive on a FAT filesystem and across a network.

- NTFS filenames allow the full character set specified by POSIX.1.

The following characters are not allowed in filenames:

\ / : * ? ! | “ < > control characters

The directory name **\$\$psxjunk** is reserved for use by the POSIX subsystem.

- Hard links work (files can have more than one name).
- Real user and group names appear as the file owner and file group when you do a long directory listing. (Note that files can be owned by a group, and not only by a user.)

On a FAT or HPFS filesystem, all files are owned by the user “Everyone” and the group “Everyone”.

You can determine the type of a filesystem by using the *df* command on a directory in that filesystem.

If you are using both INTERIX utilities and Win32 applications on the same files, keep in mind:

- Win32 applications do not provide access to case-sensitive filenames or to hard links.

- Most INTERIX utilities expect lines in script files to end in a line-feed only. The INTERIX *ed*, *ex*, and *vi* editors maintain the files as found, but some Win32 editors do not.

Converting Between POSIX and MS-DOS Text Format

Win32 applications use the MS-DOS text format, where end-of-line in a text file is marked by the characters carriage-return /line-feed. POSIX systems use the POSIX text format, where end-of-line is marked only by the character line-feed. If you open an MS-DOS text file in *vi*, each line ends with a ^M character.

Most of the INTERIX utilities require their scripts to be in POSIX format. The *ksh* and *make* utilities accept script files even when the end-of-line character is carriage-return /line-feed (the MS-DOS format) rather than line-feed (the POSIX format). The *awk* utility accepts a script (the argument to a **-f** option) in either MS-DOS or POSIX format. Other utilities require scripts to use line-feed as the end-of-line character.

The *flip* utility converts carriage-return /line-feeds to line-feeds in files. The command *flip -u* converts a text file to the POSIX format. The command *flip -m* converts a text file to the MS-DOS / Win32 format. See the *flip* reference page for more information.

INVOKING WIN32 PROGRAMS

STOPPING WIN32 PROGRAMS

AVOIDING INTERIX/WIN32 CONFLICTS

CHAPTER 4

Working with Win32 Programs

This chapter describes running Win32 programs in the INTERIX environment. You can run programs such as *cdplayer.exe* or *notepad.exe* from the INTERIX command line.

Invoking Win32 Programs

When running a Win32 program from the INTERIX environment, keep in mind the distinction between the *name* of the program to be run and the *arguments* to the program. The name must be in POSIX format; the arguments must be in the format expected by the program (usually Win32 format).

You can start the Notepad program with the command

```
//C/WINNT40/system32/notepad.exe
```

assuming your Win32 programs are stored in **//C/WINNT40/system32** and that the Notepad program file is named *notepad.exe*. Since case is important in the INTERIX system, this may not work if the program is actually named *NOTEPAD.EXE*. (Case is important if the Win32 program is installed on an NTFS filesystem.)

If you wanted to start Notepad to edit the DOS file `//C/readme.txt`, use the command

```
//C/WINNT40/system32/notepad.exe C:\\readme.txt
```

The argument, the pathname of the file **readme.txt**, is in the Win32 format. The double `\` is necessary because the INTERIX shells treat backslash as special. You could also surround the filename in single quotes, like this:

```
//C/WINNT40/system32/notepad.exe 'C:\readme.txt'
```

These techniques have some inconveniences:

- Control isn't returned to your terminal until after the Win32 program has exited.
- It's cumbersome typing the entire pathname to the Win32 program.
- You must include the file extension and remember the case of the program filename.

The first problem can be solved by running the command in the background using the **&** metacharacter:

```
//C/WINNT40/system32/notepad.exe &  
//C/WINNT40/system32/notepad.exe C:\\readme.txt &
```

The second problem can be solved in several ways:

- Add the Win32 commands to your environment by adding the appropriate directories to your *PATH* environment variable. This is the most general solution, but you still have to remember file extensions when you give the command.
- Create aliases in your environment that refer to Win32 programs. Create one alias for each program.
- Create hard links to the Win32 programs, effectively making "copies" with more convenient names. This only works if the Win32 programs are installed on an NTFS drive.

Most users find it convenient to add some directories to the *PATH* and then use aliases or links for other common Win32 commands.

Adding Directories to Your PATH

The simplest way to add the Win32 commands to your environment is to add the appropriate directories to your *PATH*. For example, if your Win32 commands are in the directory *//C/WINNT40/system32*, add this line to your **.profile** file:

```
export PATH="$PATH://C/WINNT40/system32"
```

The pathname is added in the POSIX filename format, not the Win32 format. Now all of the program files in the Windows NT **system32** directory are available from the INTERIX command line. Invoking Notepad is now a matter of typing:

```
notepad.exe
```

Aliases

In the KornShell, an alias is a command-word that the KornShell replaces with a different string. For example, the KornShell has the command *r* (for repeat) which is actually the command *fc -s*; when you type *r* as a command the KornShell substitutes *fc -s*. This makes an alias an excellent way to eliminate the need to type the extension **.exe** at the end of a command name. (C shell aliases are more like KornShell functions.)

Aliases are ideal for applications such as word processors, where the directory containing the application may have only one executable file. Adding the directory to the *PATH* clutters the environment variable to no real purpose.

Anyone can set up aliases, even without Administrator privileges.

For example, in the KornShell you can create aliases for the programs *ATTRIB.EXE* and *CACLS.EXE*:

```
$ alias attrib="//C/WINNT35/system32/ATTRIB.EXE"
$ alias cacls="//C/WINNT35/system32/CACLS.EXE"
```

creates two aliases, *attrib* and *cacls*. The command *attrib* now runs the *ATTRIB.EXE* program and the command *cacls* runs the *CACLS.EXE* program.

In the C shell, the same two aliases are created with these commands:

```
% alias attrib '//C/WINNT35/system32/ATTRIB.EXE'
% alias cacls '//C/WINNT35/system32/CACLS.EXE'
```

To create an alias for Microsoft Word in the KornShell:

```
$ alias word="//C/Program\ Files/winword/WINWORD.EXE"
```

The file **\$OPENNT_ROOT/usr/examples/win32/aliases.ksh** contains a set of KornShell aliases for all Win32 commands. The equivalent list for the C shell is stored in the file **\$OPENNT_ROOT/usr/examples/win32/aliases.csh**. (Edit these files if the names of the Win32 executables are in a different case on your machine—for example, if your system has “arp.exe” where the alias file has “ARP.EXE”.)

To include all of these aliases in your environment, you can add this line to your **.profile** (if you use the KornShell):

```
. $OPENNT_ROOT/usr/examples/win32/aliases.ksh
```

If you use the C shell, add this line to your **.cshrc** file:

```
source $OPENNT_ROOT/usr/examples/win32/aliases.csh
```

Some Win32 commands have the same names as INTERIX utilities; these have been aliased to names beginning with “w”—for example, *print.exe* is aliased to *wprint*. These commands conflict in both shells:

- *expand.exe* is aliased to *wexpand*. The Win32 command expands compressed files; the INTERIX utility expands tab characters in text files.
- *find.exe* is aliased to *wfind*. The Win32 command is equivalent to the INTERIX *grep* utility; the INTERIX utility finds files in the filesystem.
- *ftp.exe* is aliased to *wftp*. Both versions are File Transfer Protocol (FTP) clients.
- *pax.exe* is aliased to *wpax*. Both versions are implementations of the POSIX portable archive tool and both are POSIX utilities.
- *sort.exe* is aliased to *wsort*. It is a sorting utility, like the INTERIX *sort* utility, but less powerful.
- *telnet.exe* is aliased to *wtelnet*. Both *telnet* and *telnet.exe* are telnet clients.

These commands conflict if you use the KornShell:

- *fc.exe* is aliased to *wfc*. The Win32 command is equivalent to the INTERIX *diff* utility; the KornShell command displays the command history.
- *print.exe* is aliased to *wprint*. The Win32 command sends a file to the printer; the KornShell command writes its arguments to the standard output.

The Generic Windows NT System Directory

The Win32 utilities shipped with Windows NT are stored in the **system32** subdirectory of the Windows NT directory, which may be called WINNT or WINNT35 or WINNT40. The drive and directory name are stored in the environment variable *SYSTEMROOT*. You can use the INTERIX utility *ntpath2posix* to create an environment variable that contains the name of this directory. For example:

```
WINSYS=$(ntpath2posix($SYSTEMROOT)/system32)
```

If you set up this environment variable, you can replace all occurrences of **//C/WINNT35/system32** in the examples above with **\$WINSYS**.

This is a useful trick for setting up a single alias file that will be sourced by different INTERIX machines, each of which may have a different software configuration. The head of the file to be sourced looks like this:

```
WINSYS=$(ntpath2posix($SYSTEMROOT)/system32)
export WINSYS
alias append=$WINSYS/append.exe
alias attrib=$WINSYS/attrib.exe
```

Running CMD.EXE Built-In Commands

Because you can run *CMD.EXE*, the Windows NT command processor, you can also run *CMD.EXE* built-in commands. Assuming your *PATH* has been set up to include the Windows NT system directory, you could set up your INTERIX environment to give you the *dir* command for directory listings:

```
$ alias dir='CMD.EXE /c "dir /on"'
$ dir
Volume in drive C has no label.
Volume Serial Number is 14CC-6DB9
Directory of C:\USERS\stephe

08/12/96  12:12p          <DIR>          .
08/12/96  12:12p          <DIR>          ..
08/10/96  05:46p                170 .profile
07/10/96  05:41p                110 able
      . . . .
08/02/96  05:20p          <DIR>          test
07/10/96  01:33p        6,656 REPORT.DOC
```

48 File(s) 625,581 bytes
358,595,584 bytes free

Links

A link is a name for a file. On NTFS partition, files may have more than one link. Alternate names using links have the advantage of being more convenient.

For example, with the correct *PATH* these commands enable you to run the program *PING.EXE* with the command *ping*:

```
$ cd //C/WINNT35/system32
$ ln PING.EXE ping
```

Links can be useful if many users have accounts on one INTERIX machine. Links to Win32 commands can be put in a directory already in the users' default *PATH*, so no users need to change their environments.

Links have several disadvantages: your Windows NT system drive must be an NTFS partition, and you may need Administrator privileges. Experienced users also know that links can get out of hand easily and lead to an administrative nightmare.

Shell Scripts

Some Win32 commands will behave oddly when you run them from INTERIX. The reason is usually that the Win32 program examines the content of an environment variable that is already set in the INTERIX environment. The Win32 program expects the contents of the environment variable to be in Win32 format, and doesn't interpret the POSIX-format string correctly.

This doesn't apply to the *PATH* environment variable; the current value of the *PATH* environment variable is automatically flipped to the Win32 format when you run a Win32 program. This is the *only* environment variable for which this is true. If you expect your Win32 program to call *other* Win32 programs, you should make sure that the correct directories are included in the *PATH*. (See chapter 2, "Getting Started," for a brief description of the *PATH* environment variable.)

Suppose you're running a Win32 version of *emacs*. The program examines the environment for the variables *HOME* and *PATH* (among others). Because it's a Win32 program, you need to convert the values of all of the environment variables which contain pathnames except

PATH. *PATH* will be automatically converted. INTERIX provides the *posixpath2nt* command to convert pathnames. Since you don't want this change to affect other INTERIX programs, you need to run this command in a shell script.

We'll only discuss these two environment variables. Here's a sample shell script that converts only the *HOME* environment variable (*PATH* is done automatically) and runs the program *//C/win32app/emacs.exe*:

```
#!/bin/ksh
export HOME=$(posixpath2nt $HOME)
//C/win32app/emacs.exe $*
```

You can use the *#!* notation at the top of a script to invoke a Win32 program that will interpret the script. For example, suppose you had a *bash* script you wanted to run using a Win32 port of *bash*, at

D:\bin\bash.exe. This line at the top of the script would tell the POSIX subsystem that the program:

```
#!/D/bin/bash.exe
```

For Win32 programs, the pathname must be an absolute pathname.

Stopping Win32 Programs

You cannot stop a Win32 program using the INTERIX *kill* utility. The *kill* command only sends signals to INTERIX processes; it can't send a signal to a Win32 process. You can use *ps* to list any Win32 process started from an INTERIX process, but you cannot use the PID to send a signal to that process.

To stop a Win32 process, you must give the exit signal within the program (if it's interactive) or use some Win32 program such as the Task Manager to kill the program.

If you *kill* an INTERIX process which is in turn running a Win32 program, the signal is held until the Win32 program exits. After the Win32 program exits, the signal is delivered to the parent INTERIX process, which then exits.

Incidentally, although you can kill an INTERIX process using the Task Manager or *TKILL.EXE*, you shouldn't. When an INTERIX process is killed from the Win32 environment, certain important clean-up functions aren't performed. The results (especially if you make a habit of it) are likely to be both subtle and bad.

File Redirection

Win32 programs can accept input redirected from standard in (as in `< file`); you can also redirect their standard output (with `> file`) and their standard error.

Win32 programs can also be used in command pipelines. But some Win32 programs are not well-behaved; they “poison” any pipeline they feed into. In order to provide more robust behavior when piping to and from Win32 programs, INTERIX includes a Win32 utility, *cat32.exe*, which is simply a “better behaved” filter. For example, If you experience problems with a particular Win32 program in a pipeline, try inserting *cat32* into the pipeline. For example:

```
$ net.exe users | cat32 | more
```

Avoiding INTERIX/Win32 Conflicts

The best way to avoid problems is by planning ahead.

When setting up a user in the User Manager:

- Assign a home directory other than `//C/USERS/default`.
- Set the ownership of the home directory to the user. (A Win32 application may have already set the ownership to something unexpected.)
- Assign a reasonable owner and group to the home directory; all files and subdirectories created will inherit ownerships properly.

Similar rules apply to creating project trees. The root of the tree should have a reasonable group ownership; all subdirectories and files will inherit that group.

Depending on your needs, you may find it convenient or useful to segregate INTERIX and Win32 files in different directories—for example, all MS Word files may be in their own subdirectory of the documents directory.

If you want to see the most granular view of ACEs on a file, run the Win32 *CACLS.EXE* utility. It lists each ACE on a file. For a complete discussion of ACLs and ACEs, including the most basic building block ACEs visible with *CACLS.EXE*, see the discussion of security in the *Windows NT Resource Kit*, Volume 1.

Avoid using filenames in the POSIX environment that are not allowed in the Win32 environment. For example, files in Win32 cannot have the same basename as a Win32 device: **aux**, **com1**, **com2**, **com3**, **com4**, **lpt1**, **nul**, and **prn** will not be recognized by a Win32 program, regardless of the file extension.

For example, if you are using the Microsoft Visual C / C++ compiler with the INTERIX Software Development Kit, you must rename any files with names such as “aux.c”. Because the compiler is a Win32 program, it does not recognize these filenames.

THE INTERIX TERMINAL

ENABLING CONSOLE PROPERTIES

ALTERNATE TERMINAL WINDOW SIZES

VIRTUAL DEVICES

TAPE DRIVES

CHAPTER 5

Terminals and Other Devices

This chapter describes the INTERIX terminal console window and some of the virtual devices, including the tape device.

The terminal is the console window displayed when you run an INTERIX shell. You can resize it, change the colors, and scroll back through previously-displayed material. The terminal window is controlled both through the *stty* command and by adjusting the Windows NT properties of the console window.

The INTERIX Terminal

INTERIX uses a console window as a POSIX-style character terminal. The terminal type is named `opennt`, described in the database **\$OPENNT_ROOT/usr/share/terminfo**. (In releases before 2.0, the terminal type was named “NTconsole”, but this name doesn’t conform to the terminfo terminal naming conventions.) The utilities *vi*, *more*, *clear* and the shell use this file or the **termcap** file. **\$OPENNT_ROOT/usr/share/termcap** file.

The default settings for end-of-file and other control characters are:

End-of-File	^D
Interrupt	^C
Kill	^U

Suspend ^Z

You can change these settings with the *stty* command.

If you are logged into a system which does not have the *opennt* terminal type defined (using the *telnet* client, for example), set **TERM=ansi**.

You might want to copy the *opennt* terminal description from the **\$OPENNT_ROOT/usr/share/termcap** file to the remote system. The *opennt* terminal type is functionally like an *xterm* but the key sequences logged in **termcap** and **terminfo** more closely resemble a *vt100*.

You can also redirect to and from **/dev/null** and **/dev/tty**. The name **/dev/tty** always refers to the current terminal. The *tty* command reports the name of your current terminal. You can also redirect information between terminal windows; if you have the terminals **/dev/ttyn00** and **/dev/ttyn01** open, you can redirect output from a command running in **/dev/ttyn00** so it appears in **/dev/ttyn01**.

Enabling Console Properties

The INTERIX shells or terminal windows run in console windows. There are several features you may want to enable on your INTERIX terminal windows:

Windows NT cut and paste

Select, copy or cut, and paste text using the mouse and the Windows clipboard buffer.

Screen colors

Set the colors used for the screen background, text, and highlighted text.

Scroll bars Display text that has already scrolled off the top of the screen.

Title bars Change the text of the terminal window title bar; you might want to display the value of the current working directory or of the currently-executing command.

The next sections describe how to enable these features. The properties of the console windows are set by Windows NT; therefore, changing most of these properties is done through Windows NT.

Enabling Windows NT Cut and Paste

The INTERIX shell runs in a console window. To set up normal Windows NT cut-and-paste in an open console window:

1. Open the window menu, either by clicking on the upper left corner or by pressing **Alt-Spacebar**.
2. Select the **Properties** item.
3. On Windows NT 3.51, select the **General Settings** tab; on Windows NT 4.0, select the **Options** tab.
4. Select the **Quick Edit Mode** box.
5. Click on **Save**. If you want, you may configure the option to take effect for all INTERIX shell windows, when prompted.

Once cut and paste is configured:

- Left mouse button drag to select text in the shell window
- Click right mouse button once to cut the selected text to the clipboard.
- Click the right mouse button again to paste the captured text from the clipboard.

Setting Screen Colors

You can set the colors for your INTERIX terminal windows by changing the properties of your INTERIX icon (Windows NT 3.51) or shortcut (Windows NT 4.0).

Colors for screen windows are picked from a palette of 16 screen colors. They are two sets of eight: the first eight colors are dark shades of the second eight colors. If color 3 is a dark green, color 11 should be a bright green. Text highlighting uses the color that corresponds to the text color—if text is displayed in color 3, then highlighted text is in color 11. If text is displayed in color 11, highlighted text is in color 3.

You can change the colors in the palette. If you change one color, you should also change the corresponding color.

To set the colors under Windows NT 3.51:

1. Select the INTERIX shell icon.
2. Open the window menu, either by clicking on the icon in the upper left corner or by pressing **Alt-Spacebar**.

3. Select the **Properties** item from the menu to display the Properties box.
4. Select the **Colors** tab.

You can set a color for your Screen Text and your Screen Background. The settings for Popup Text and Popup Background have no effect on an INTERIX Terminal window.

5. Select the *Screen Text* button and then click on the color you want used to display text in the terminal window.
6. Select the *Screen Background* button and then click on the color you want used to display the terminal background.
7. When you're satisfied with your selections, click on the *OK* button.

Windows NT displays a dialog box asking whether you want the changes to affect only the current window or all windows started from this icon or shortcut.

8. Select "all windows" and click on the *OK* button.

You may want to exit the shell window and start a new one; when you change the colors in an open shell, some of the utilities (such as *more*) reset the colors to their previous values.

To set the colors under Windows NT 4.0:

1. Select the INTERIX shell shortcut.
2. Right-click to display the shortcut menu.
3. Select **Properties** from the menu to display the Properties box.
4. Select the **Colors** tab.

On the Colors properties tab, you can set a color for your Screen Text and your Screen Background. (The settings for Popup Text and Popup Background have no effect on an INTERIX Terminal window.)

5. Select the *Screen Text* button and then click on the color you want used to display text in the terminal window.
6. Select the *Screen Background* button and then click on the color you want used to display the terminal background.
7. When you're satisfied with your selections, click on the *OK* button.

Windows NT displays a dialog box asking whether you want the changes to affect only the current window or all windows started from this icon or shortcut.

8. Select “all windows” and click on the *OK* button.

Enabling Scroll Bars

You can also enable Windows NT scroll bars on your terminal windows. Scroll bars let you retrieve previously-displayed text.

To enable scroll bars, enlarge the buffer size for the terminal window. To do this:

1. Open an INTERIX terminal window.
2. Open the window menu, either by clicking on the icon in the upper left corner or by pressing **Alt-Spacebar**.
3. Select the **Properties** item from the menu to display the Properties box.
4. Select the **Layout** tab.
5. In the Screen Buffer Size box, enter into the Height field the number of lines of text you want stored. The default is 25, one screenful. Most people want to save at least 200 lines of text, but the more text you save, the more memory you use.
6. Click on the *OK* button.

Windows NT displays a dialog box asking whether you want the changes to affect only the current window or all windows started from this icon or shortcut.

7. To make the changes affect subsequent windows:
 - On Windows NT 3.51, select the button labelled “Save properties for future windows with the same title”.
 - On Windows NT 4.0, select the button labelled “Modify shortcut which started this window”.
8. Click on the *OK* button.
9. Exit the window and start a new terminal window. The change does not take effect until you do this.

Changing the Terminal Title Bar

You can specify the text displayed in the title bar of an INTERIX terminal window. This is often used to display the currently-executing command in the title bar.

If your terminal type is anything other than an INTERIX console window or an *xterm* window, you should turn off the display of commands in the title bar.

The title string can be up to 255 characters long; the title is displayed as proportional text, so you may want to use double spaces to separate words.

The title is set using the escape sequence `\0033]0;text\a`, where `\0033` is the octal code for the escape character and `\a` is the escape sequence for the alert, or control-G. You can also send the escape sequence by using the command control-V Escape, and the control-G sequence by using control-V control-G. To echo the string to the terminal, use **echo -n**.

For example, to set the terminal title bar to “Telnet link to Gandalf”:

```
echo -n "\0033]0;Telnet link to Gandalf\a"
```

This sequence is similar to the sequence for setting the title bar in X Windows, but doesn’t behave quite the same. In X Windows, the window and the icon can have different titles, but in INTERIX, the icon has the same title as the window.

For example, if you want to display your current directory in the terminal title bar, you can do it in the KornShell by setting this code in your **.profile** or **.enviro**n file:

```
_cd () {
    if test $# -eq 0
    then
        cd
        echo -n "\033];$PWD\a"
    elif cd "$*"
    then
        echo -n "\033];$PWD\a"
    fi
}
alias cd="_cd"
```

The alias ensures that when you type *cd*, it’s the *_cd* function that’s executed. The function says that if the *cd* command succeeds, the

terminal bar should be set to the value of the current working directory, stored in the *PWD* shell parameter.

If you want to display the current command and you are using the KornShell, you can set the **cmd_intitle** option with **set -o**:

```
set -o cmd_intitle
```

In the C shell, the shell variable *showcmd* controls the display of the command in the title bar; if *showcmd* is set, then all non built-in commands are displayed:

```
set showcmd
```

Alternate Terminal Window Sizes

The default terminal window (specified by the terminal type *opennt*) gives a console window that is 80 characters wide by 25 characters high. You can use *stty* to change the terminal size (recommended) or you can reconfigure your terminal window's properties and set the *TERM* environment variable to an appropriate terminal description.

To change the terminal size using *stty*, use the *rows* and *cols* operands. For example, to make the terminal window 80 columns by 50 rows:

```
$ stty cols 80 rows 50
```

To make the terminal window 125 columns by 50 rows:

```
$ stty cols 125 rows 50
```

The width is specified as 125 rather than the traditional 132 because 125 columns of the standard Windows NT terminal font fills the screen.

You can place the *stty* command in your startup file.

You could also set your *TERM* environment variable to the alternate terminal description and alter the terminal window properties. The available terminal types are:

opennt-35 A window 80 columns by 35 rows.

opennt-50 A window 80 columns by 50 rows.

opennt-60 A window 80 columns by 60 rows.

opennt-*-w-*

Versions of the above terminal heights also exist with names including **-w**; these are 125 columns wide instead of 80.

opennt*-nti

Versions of the terminal types exist with names ending in **-nti**. These are versions which have no **ti/te** (terminal initialization/terminal end) capability. If you set *TERM* to one of these, programs such as *vi* and *more* will not save the screen when you begin the program and restore it when you exit—the program session is not erased from the screen when you exit the program. This behavior is more like the default behavior on System V machines.

The *TERM* environment variable is usually set in your **.profile** or your **.cshrc** file in your home directory.

To alter the terminal window description:

1. Open an INTERIX terminal window.
2. Open the window menu, either by clicking on the icon in the upper left corner or by pressing **Alt-Spacebar**.
3. Select the **Properties** item from the menu to display the Properties box.
4. Select the **Layout** tab.
5. In the Window Size box, enter the dimensions of the new terminal window. For example, if you want to use the **opennt-60** terminal, set the *Height* field to 60.
6. Click on the OK button.
Windows NT displays a dialog box asking whether you want the changes to affect only the current window or all windows started from this icon or shortcut.
7. To make the changes affect subsequent windows:
 - On Windows NT 3.51, select the button labelled “Save properties for future windows with the same title”.
 - On Windows NT 4.0, select the button labelled “Modify shortcut which started this window”.
8. Click on the OK button.
9. Exit the window and start a new terminal window.

Virtual Devices

A number of the devices in the INTERIX subsystem are virtual devices: they do not have an implementation in the filesystem. The **/dev/null** device, the ttys (including **/dev/tty**) and the pseudo terminals, and the tape drives. The command **ls /dev** lists the contents of the virtual device space.

The virtual devices are:

/dev/ntape[0-9]

Tape drives opened in non-rewind mode.

/dev/null

The null device.

/dev/proc

The process list (used by *ps*).

/dev/ptmx

Master pseudo terminal, in the System V manner.

/dev/pty[p-zA-E][0-9a-f]

Master pseudo terminals, in the Berkeley manner.

/dev/tape[0-9]

Tape drives opened in rewind mode.

/dev/tty[p-zA-E][0-9a-f]

Slave pseudo terminals.

/dev/tty

The current terminal.

/dev/ttyn00-/dev/ttyn63

Terminal windows.

The pseudo terminals are used by different utilities, including *xterm* and *telnet*. INTERIX Release 2.2 allows 256 pseudo-terminal connections in total. A pseudo terminal being used by *xterm* is not available for use in a *telnet* session.

Tape Drives

INTERIX supports the same tape devices that Windows NT supports; the actual tape driver software is part of Windows NT, not INTERIX. INTERIX provides access to that tape device through the INTERIX subsystem. The utilities *tar*, *pax*, *cpio*, and *dd* “understand” the tape devices. The *mt* utility gives commands to the tape drive device.

The *name* of the tape device controls the mode in which the device is opened. The devices **/dev/tape0** and **/dev/ntape0** refer to the same device, but the former is opened in rewind mode and the latter is opened in non-rewind mode.

Note that a tape must be in the drive before utilities will be able to read or write from the tape.

Tape Portability

While the tar file format is a very portable format for data exchange, using *tar* with tape devices can be a frustrating experience. This is because the *file* format is portable, but the *tape* format is often not.

Operating systems access tape devices in many different ways and the default behavior on one platform is probably not the default behavior on another platform. This incompatibility usually shows itself in two ways; blocking factors and hardware compression.

Blocking Factors

Blocking refers to the size of a “block” of data as written to the tape device. When data is written to a tape device with a particular tape block size, then the tape must be read with the same size of tape block. There are a number of common block sizes that may be found when using different operating systems: 512, 1024, 10240 and “variable”. For example, System V-style systems often default to a blocking factor of 512, while BSD-style systems default to a blocking factor of “variable”. The blocking factor used when writing a tape depends upon the configuration of the system, any commands issued to the tape drive using *mt*, and the blocking factor specified in the command which wrote to the tape drive (usually *tar*).

If you don't know the size of the block that was used when the tape was written, you may have considerable difficulty when attempting to read the tape on INTERIX (or any other platform).

When creating a tar archive on a non-INTERIX system, you can save yourself time by specifying the blocking factor with the *dd* command. For example, when creating an archive on an SVR4 system, this command ensures a block size of 10240:

```
$ tar cvf - . | dd of=/dev/tape obs=10240
```

The **of** specifies the output file and the **obs** specifies the block size. By using *dd*, you sidestep any system-specific values for *tar*.

But it may be necessary to set the tape device block size. Different systems use different commands. You may first need to set the tape device block size with the *tapecntl* or *mt* program and then create your tar archive. With *tapecntl* (available on SVR4 and other systems):

```
$ tapecntl -f 10240
$ tar cvf - . | dd of=/dev/tape obs=10240
```

This tape is now readable by INTERIX in nearly every circumstance with the following command:

```
$ dd if=/dev/tape0 ibs=10240 | tar xvf -
```

However, on INTERIX, you may still need to set the Windows NT tape device to the proper block size, and then read the tape. The *mt* subcommand **setblk** sets the block size:

```
$ mt setblk 10240
$ dd if=/dev/tape0 ibs=10240 | tar xvf -
```

To read a tape that was created on an INTERIX system on another system, use the same commands to set the tape device block size. For example:

```
$ tapecntl -b 10240
or
$ mt setblk 10240
$ dd if=/dev/tape ibs=10240 | tar xvt -
```

Hardware Compression

Some tape devices, most notably those on HP-UX, have devices that provide data compression via specialized hardware. You should avoid using the compression tape devices since this hardware may not exist on the INTERIX platform. Without the hardware, it will not be possible to read these tapes when hardware compression is used. These devices are designated specific to the system; if you don't know the name of these devices, ask your system administrator.

When using the non-hardware compression device, you can follow the steps as above in “Blocking Factors”.

For more information about manipulating tape drives, see the *mt* reference page.

OVERVIEW

THE CRON SERVICE

ADMINISTERING OTHER SERVICES

THE TELNET DAEMON

THE SYSLOG DAEMON

CHAPTER 6

Services and Daemons

On many operating systems, repetitive system services are handled by server programs, usually called “daemons.” The advantage of a daemon or a service is that no user has to be logged in for the task to be done.

Windows NT provides its own server-level programs, and INTERIX takes advantage of those services to provide services such as *cron*, which runs programs according to the schedule set by the *crontab* utility, and *telnetd*, which allows users to login over the network.

INTERIX provides two programs to administer INTERIX services: *cronadmin* and *service*. You must have Administrator privileges to run either of these programs.

- The *cronadmin* program installs a *cron* service for a user. Once *cronadmin* has been used to install a *cron* service for a user, the user can schedule jobs using the *crontab* command.
- The *service* program registers, installs, starts and stops an INTERIX program running as a service.

Services make use of the program *psxrun.exe*, which INTERIX installs into the Windows NT **system32** directory.

Three daemons-as-services are provided: *cron* (with *at* and *batch*), *telnetd*, and *syslogd*.

Overview

The Windows NT system provides a Service Control Manager which handles services. Services are either automatic (are begun on system startup) or manual (must be started by the user).

The **Services** applet in the Control Panel is one way of controlling services on Windows NT; using the **Services** applet, you can add, remove, view, start, and stop services. For security reasons, each running service is affiliated with a user and a password and has its own unique ID number. Most services are associated with the user “Local System” which is provided for that purpose and doesn’t require a password. Information on each service is available in the Registry, but it’s normally invisible to the user.

In order to run a service, your user account must have the “Service Logon Right” privilege. This privilege is automatically given to your account the first time you start or install a service using the *INTERIX service* utility. It is possible for the Administrator to remove that right from your account, however, and then you wouldn’t be able to run services.

A service has no console display. You cannot send a signal to a service using *kill*, though you can stop it using the **Services** applet of the Control Panel or using the *service* or *cronadmin* utilities.

A service runs in a minimal environment. It has only the environment assigned to the system’s “Default User” (see the **System** applet of the Control Panel), plus the *TZ* environment variable. Its standard input, output, and error are all redirected to **/dev/null**.

The Cron Service

The *cron* service runs commands according to a schedule. Each user gets a personal *cron* service, but this must be enabled with the *cronadmin* utility, by someone with Administrator privileges. You add, delete, and edit commands run by *cron* using the *crontab* utility.

Cron is different from most other services because *cron* provides a more extensive environment and mails you the standard output and standard error of the command. A *cron* job sets the following environment variables in the execution environment: *HOME* (your home directory), *LOGNAME* (your login name), *PATH* (a *PATH* that is guaranteed to

find the INTERIX utilities, though it won't necessarily be your *PATH*), and *SHELL* (the pathname to *sh*).

Setting Up Cron

Before you can run commands using *cron*, it must be enabled for you. Since *cron* needs your password, you should be there. The *cron* service is enabled by someone with Administrator privileges on your machine, using the *cronadmin* program. The *cronadmin* program is a command-line control interface that administers the *cron* service. It has the following subcommands:

add user password	Adds <i>user</i> to the list of people with a <i>cron</i> service; <i>password</i> is the user's password. If the password is not provided (and it usually isn't), <i>cronadmin</i> prompts for the password.
grant user	Grants the "Service Logon Right" to <i>user</i> .
list	Lists all <i>cron</i> services on this machine.
remove user	Stops <i>user</i> 's <i>cron</i> service and uninstalls it.
revoke user	Revokes <i>user</i> 's "Service Logon Right".
start user	Starts the <i>cron</i> service for <i>user</i> .
stop user	Stops (kills) the <i>cron</i> service for <i>user</i> .

Anyone can run **cronadmin list**; the other subcommands require you to have Administrator privileges.

Scheduling Jobs Using Crontab

Once your *cron* service is enabled, you can schedule commands using the *crontab* utility. *Crontab* maintains the table of *cron* entries (hence the name); to edit your schedule, give the command **crontab -e**. The *crontab* utility starts up an editor. Your schedule is stored in your *crontab* file, which is **\$OPENNT_ROOT/usr/lib/cron/\$LOGNAME**

A *crontab* entry is the time or times when the command should be run and the command itself, in this format:

minute hour day_month month day_week command_string

Minute is a numeric value from 0-59.

Hour is a numeric value from 0-23.

Day_month is a numeric value from 0-31, which indicates the day of the month.

Month is a numeric value from 1-12.

Day_week is a numeric value from 0-6, where 0 is Sunday.

The rest of the line is taken as the command to be run.

The numeric value can be given as a number, as a range (two numbers separated by a hyphen), as a comma-separated list, or as an asterisk, which represents all values.

For example, to run a shell script named *progress* at 15 minutes past the hour, every hour of every day, use the crontab entry

```
15 * * * * progress
```

To run *progress* at 5 AM and 9 PM on Monday through Thursday:

```
0 5,9 * * 1-4 progress
```

When you save the file and exit the editor, the entry is added to your schedule.

You can see your current schedule with the command **crontab -l** (that's the letter ell).

Scheduling Jobs Using At and Batch

Two other facilities exist for scheduling jobs, but they both rely on the *cron* service. Before you use either *at* (run a job at a particular time) or *batch* (run a batch of jobs), you must have an Administrator set up the *cron* service for your account.

The *at* command runs a job or jobs at a later time. Unlike *crontab*, which lets you schedule a job for regular intervals, *at* schedules a single execution of a job. *At* is useful for running a particular process at midnight, for example, when some system resources are free.

To run a job with *at*, type the command and the time when you want the job to run. Then type the commands, one to a line, and then type Control-D to end input.

For example, to run a script called *printoff* at 11:00 PM today, type

```
$ at 2300
printoff
^D
at271889b
```

The *at* command schedules the job for execution at 2300 (11:00 PM). The time is given in 24-hour format. The number output is the ID

number of the job; you can use the ID number of the *at* job to kill remove the command from the schedule.

To remove a command, use the **-r** option:

```
at -r at_job_id
```

To list all of the *at* jobs you currently have scheduled, use the **-l** option.

An *at* job may not run at precisely the time specified. The *at* jobs are actually carried out by a command called *atrun*; the *atrun* command runs every 10 minutes. (If you want *atrun* to run more or less frequently, include the command *atrun* in your **crontab** file.)

The *batch* command is like specifying *at* with the time “now”. The job will actually run the next time that *atrun* runs.

Administering Other Services

The INTERIX Commands and Utilities uses a utility called *service* as a command interface to the service manager. It’s an INTERIX command-line utility that controls the service manager, just as the **Services** panel in the Control Panel does. You can run any INTERIX command line as a service. (Do not run Win32 programs through this interface; the windows won’t appear, and it won’t be possible to kill the Win32 program.)

The *service* utility is a more general utility than *cronadmin*. Using *service*, you can:

- List all services running; anyone can do this.
- Grant the “Service Logon Right” to a user. You must have Administrator privilege to do this.
- Revoke the “Service Logon Right” from a user. You must have Administrator privilege to do this.
- Start a service. You must have Administrator privilege to do this.
- Stop (kill) a service. You must have Administrator privilege to do this.
- Remove a user’s service; this kills the service and “uninstalls” it. The user still has the right to start another service. You must have Administrator privilege to do this.

With the *service* utility, any command can be run as a service, though most commands are not well-suited to being services. (Most commands exit, for example, or try to make use of standard input, output or error, or try to display information.) The best commands to run as services using *service* are ports of daemons.

For network daemons (such as *rlogind*), run them under the control of the *inetd* service.

The Telnet Daemon

The telnet daemon *telnetd* is provided as part of the INTERIX Workstation and the INTERIX Server products.

When you are running the telnet daemon on your machine, other users can login to your machine (using a telnet client), if they have accounts on your machine.

Installing the Telnet Daemon

If you already have a telnet service installed (for example, the telnet service supplied with earlier releases of INTERIX), you must uninstall it before installing *telnetd*.

To install the telnet daemon as a service:

1. Login to the Administrator account.
2. From an INTERIX shell, install the service using the *service* command:

```
$ service install //C/Interix/bin/telnetd
```

3. Once the service is installed, you must start it:

```
$ service start telnetd
```

To stop the service, login as Administrator and use the command

```
$ service stop telnetd
```

To remove the service, login as Administrator and use the command

```
$ service remove telnetd
```

The service can also be administered through the **Services** applet of the Windows NT Control Panel.

Using Telnet and Telnetd

When a user logs into your system using *telnet*, the default shell is **/bin/sh**.

The INTERIX *telnetd* requires users to login using their Windows NT domain and their login name. Login names are case-sensitive, and must be entered as they were entered into the User Manager. If the user doesn't enter a domain name, a default is assumed. (The *telnetd* displays the default domain in the banner it presents to users.)

To change the default domain name assumed for *telnet* logins, set the value of the registry entry **HKEY_LOCAL_MACHINE | SOFTWARE | Softway Systems | OpenNT | LoginDefaultDomain** to the name of the default domain.

If the *telnetd* is running on a Windows NT workgroup server which has no domains, set the value of the **LoginDefaultDomain** registry entry to be a null string. If you do not do this, users must specify an empty domain name in their *telnet* login name, like this: *+loginname*.

The Syslog Daemon

The syslog daemon, *syslogd*, is a facility for storing and rerouting log messages from applications and system services. The *syslogd* facility is administrative: when something goes wrong, you can check the log files for any relevant messages, or have them written to your terminal screen or to another computer.

Syslogd handles only INTERIX processes which were written to use the *syslog()* API; it does not handle log messages from the Win32 subsystem. (For those, use the Windows NT Event Log and Event Viewer.)

If *syslogd* is not running as a service, messages intended for *syslogd* are all appended to the file **\$OPENNT_ROOT/var/adm/log/logger**.

Note: This file, like all log files, can grow very large. Check it regularly and shorten or delete it when it grows too large.

To run *syslogd* as a service, first install it using *service*:

1. Login to the Administrator account.
2. From an INTERIX shell, install the service using the *service* command. *Syslogd* must have the **-s** option:

```
$ service install //C/Interix/bin/syslogd -s
```

3. Once the service is installed, you must start it:

```
$ service start syslogd
```

The *syslogd* service is configured with the file **syslog.conf** in the **\$OPENNT_ROOT/etc/** directory.

The syslog.conf File

The **syslog.conf** file specifies which system log messages will be stored and where they will be stored.

Each line of the file contains either a comment (comments are lines whose first printing character is #) or a specification. A specification has two parts, the *selector* (the messages affected by this line) and the *action* (the action to take when those messages show up). The selector and the action are separated by one or more tab characters.

The selector specifies which messages are affected, based on the facility which sent the message, and the severity or level of the message.

The facilities are:

auth	The user authorization system, such as <i>login</i> .
authpriv	The authorization system (as auth), but the log file can be read only by privileged users.
cron	The <i>cron</i> scheduling daemon.
daemon	System daemons not explicitly provided for by other facilities.
ftp	The file transfer protocol (FTP) and associated utilities.
lpr	The line printer spooling system.
mail	The mail system.
mark	This is a special facility which receives a message of priority “info” from <i>syslogd</i> every 20 minutes. See the <i>syslogd</i> reference page for more information.
news	The network news system.
syslog	The <i>syslogd</i> daemon itself.
user	Messages generated by other user processes. This is the default if no other facility is specified.
uucp	The UUCP message transport system.

local0—local7

These 8 facilities are reserved for local use.

An asterisk (*) character specifies *all* facilities except **mark**.

The levels are, in order of decreasing seriousness:

emerg	A panic condition.
alert	A serious condition that should be corrected immediately, such as a corrupted database.
crit	Critical problems, such as device errors.
err	System errors.
warning	Warnings
notice	Conditions that should be handled specially but that aren't errors.
info	Informational messages.
debug	Messages that are normally only useful when debugging a program.

The *action* field specifies what *syslogd* should do when the specified facility has a message of the specified level or higher. There are four possible actions:

- A *filename*. The messages are stored in this file. The *filename* begins with a leading slash and is relative to the **\$OPENNT_ROOT** directory. For example, the filename **/var/adm/tse** refers to the file **\$OPENNT_ROOT/var/adm/tse**. Filenames beginning with **/dev** refer to a particular device. The **/dev/console** device doesn't exist on INTERIX.

Note: The directory must already exist; *syslogd* will not create directories.

- A *hostname*. Messages are sent to *syslogd* on the machine *hostname*. The *hostname* is a machine on the network, and its name is preceded by an @ sign. For example, to send messages to *syslogd* on the host "aztec", use the hostname "@aztec".
- A comma-separated list of *users*. Messages are written to the listed users, if they are logged in. Remember that user names are case-sensitive, and must include the domain.
- An *asterisk*. Messages are written to all users.

Here are some sample lines from a **syslog.conf** file:

```
mail.*                /var/adm/log/mail
*.info                @ramses
*.info                /var/log/messages
*.emerg               *
```

The first line logs all mail messages in the file **\$OPENNT_ROOT/var/adm/log/mail**. The second sends all messages of level “info” and higher to the machine “ramses”. The third sends those same messages to the file **\$OPENNT_ROOT/var/log/messages**. Notice the same selector can be specified more than once. The fourth line sends all messages of level “emerg” to all users.

THE ON-LINE LIBRARY

THE REFERENCE PAGES

THE MAN COMMAND

THE ADOBE READER

TECHNICAL NOTES

CHAPTER 7

Other Documentation

This chapter describes the documentation available with INTERIX.

INTERIX provides documentation in two forms: as Portable Document Format (PDF) files, which can be viewed on-line or printed with the Adobe Acrobat Reader 3.0 program, and as formatted text files, suitable for reading using the *man* utility.

All of the INTERIX documentation is available in the form of PDF files. **To read the on-line documents you must have the Acrobat reader installed.** A distribution of the Acrobat reader is provided with the INTERIX CD-ROM.

The On-Line Library contains all of the PDF files for all INTERIX products. They are installed on your system in the directory **\$OPENNT_ROOT/pubs** if you chose the INTERIX PDF Documentation option when installing. Otherwise you can read them from the INTERIX Media Kit CD-ROM, in the **/PUBS** directory.

The On-Line Library

The INTERIX On-Line Library contains these documents as PDF files.

- The *Getting Started* booklet in PDF and Microsoft Word formats, **GETSTART.PDF** and **GETSTART.DOC**.

- This User's Guide, **CUINTRO.PDF**.
- The *Software Development Kit User's Guide*, **SDKINTRO.PDF**.
- The *INTERIX Commands and Utilities Reference*, **CUREF.PDF**.
- The *Elvis Manual*, **ELVIS.PDF**. The *elvis* editor is included as an alternative to *nvi*; it is a *vi* clone.
- The X11 server documentation. More information about this documentation can be found in the file **\READ_X11.TXT** on the INTERIX CD-ROM.
- The system interface and library references: **SIREF.PDF**, the system interfaces and basic C library reference pages; **CURSES.PDF**, the Curses documents; **XLIB.PDF**, the X library routines reference pages; and **MISCINFO.PDF**, general system information and file format descriptions.
- Legal information, such as the copyright document in PDF and text formats, **CPYRIGHT.PDF** and **CPYRIGHT.TXT**; a letter outlining the Year 2000 policy for INTERIX, the Free Software Foundation's General Public License, **GPL.TXT**, and Library General Public License, **GLPL.TXT**.
- POSIX.1 and POSIX.2 Conformance Documents, **PCD1.PDF** and **PCD2.PDF**. These documents describe the INTERIX behaviors that are required by the POSIX.1 and POSIX.2 standards to be documented.

There may be other documents as well.

The Reference Pages

The reference pages for the utilities are available on-line both as a PDF file, which you can read using the Acrobat Reader, and as formatted man pages, which you can read using the *man* utility. If you have logged into the INTERIX system using *telnet*, you must use the *man* utility, since the Acrobat reader is a Windows application.

Utility reference pages which apply only to the INTERIX Software Development Kit have the header "SDK".

The Man Command

INTERIX also provides the *man* utility. The *man* command displays the reference page for a given command or topic.

The *man* command searches the directories listed in the *MANPATH* environment variable; if *MANPATH* is not set, *man* searches the directory **usr/share/man** under the **\$OPENNT_ROOT** directory. Within each searched directory, *man* looks for directories named **catx**, where *x* is a string, usually a digit. INTERIX uses the following subsections for the manual sections:

- 1 Utility reference pages
- 2 System interfaces
- 3 Library calls
- 4 Device descriptions
- 5 File formats and other miscellaneous information

The command **man printf** retrieves the *printf* reference page from section 1; the command **man 3 printf** retrieves the *printf()* reference pages from section 3.

If you created a directory named **catadmin** which also contained a *printf* reference page, the command **man admin printf** would retrieve that reference page.

The default setting for the *MANPATH* environment variable is (line break inserted for readability):

```
$OPENNT_ROOT/usr/share/man:  
$OPENNT_ROOT/usr/X11R5/man
```

You can override the value of the *MANPATH* environment variable and search other directories by specifying them on the command line with the **-M** option.

The reference pages are flat ASCII files. Emphasis and underlines are done with backspace (^H) sequences: a bold letter is repeated, so **b^Hb** displays as an emphasized “b”. An underlined letter is followed by a backspace and an underscore: **b^H_** also displays as an emphasized “b”. Future enhancements to the INTERIX terminal interface may change this behavior.

Searching for Keywords

If you do not know which reference page might address a particular topic, you can also use the *man* command to search the subject lines for each reference page:

```
man -k topic
```

The *topic* is a basic regular expression. The *man* command performs a case-insensitive search for subject lines containing *topic*. On many systems, the command *apropos* is an alias for **man -k**.

The Acrobat Reader

The INTERIX documentation is also shipped in Adobe Portable Document Format (PDF); these files are read using the Adobe Acrobat reader.

The advantage of the Acrobat reader over the traditional *man* command is that the Acrobat reader gives a complete printing environment, including formatting, for off-line reading. It also provides hypertext links connecting the reference pages (in the “SEE ALSO” sections) and a bookmarks list that is also a table of contents or an alphabetical listing of reference pages.

The PDF files are stored in the directory **\$OPENNT_ROOT/pubs**.

Viewing on a 640x480 Display

The initial reader setup is for a display supporting 1024x768. If you are viewing these files on a 640x480 display, reading will be easier if you:

- Turn off bookmarks by clicking on the upper left button on the tool bar.
- Change the magnification to about 125% using the magnification menu at the bottom of the screen.
- Choose Fit Visible or Fit Width from the View menu, depending upon your preference.

Technical Notes

Softway Systems also provides technical notes. The technical notes cover specialized topics or provide information in a more timely manner than the *User's Guide*. The technical notes are available from the INTERIX World Wide Web site, [**http://www.interix.com/**](http://www.interix.com/).

SETTING UP X11 APPLICATIONS

RUNNING X11 APPLICATIONS

FURTHER READING

CHAPTER 8

X Window Clients

INTERIX Release 2.2 contains X11 client programs such as *xterm*, *xcalc*, *xlsfonts*, and *xrdb*. These X client programs were compiled with the X11R5 libraries, but will run with an X11R5 or an X11R6 server.

Setting Up X11 Applications

Before you can run any of the INTERIX X11 clients on your machine, you must have an X11 server installed and running on your INTERIX machine, and you must set the *DISPLAY* environment variable.

For information about the X11 server provided with INTERIX Workstation and Server, see the file **\READ_X11.TXT** on the INTERIX CD-ROM. It will point you to other documentation for the X11 server.

To set the *DISPLAY* environment variable, which is used by the X clients, edit your **.profile** file to include the lines

```
export DISPLAY=localhost:0
```

If you are using the C shell, edit your **.login** or **.cshrc** file to include the lines

```
setenv DISPLAY localhost:0
```

The environment variables will be set in your shell the next time you start an INTERIX console window.

The INTERIX default startup also sets up your *PATH* environment variable to include the directory **\$OPENNT_ROOT/usr/X11R5/bin**, which is where INTERIX installs the X client binaries. It also sets the *XAPPLRESDIR* environment variable, which is used by X clients to locate their resources.

Running X11 Applications

To run an X11 application, type the command line at the INTERIX terminal prompt. For example, to run the *xclock* program, type the command

```
xclock
```

It's useful to run X11 applications in the background so you have access to the terminal prompt. To run *xclock* in the background, type the command

```
xclock &
```

How the X clients are displayed depends upon how you have configured the X11 server and what window manager you are running. Most X11 servers running under Windows NT have two display modes:

Single-window mode: One application window is opened as the X root window, and all X clients exist in this window. You must start a window manager (such as *twm* or *mwm*).

Multiple-window mode: Each X client is opened in an individual window, controlled by the Microsoft Windows window manager or an X Window window manager, if you start one.

The *twm* window manager is one of the X11 clients included with INTERIX. It can be run from the command line or from a desktop shortcut, as described in chapter 3.

Many window managers (such as *twm* and *mwm*) provide root window menus, typically available by clicking the left mouse button on any empty section of the screen. Before you can display the root window menu, you must enable the root window. Once you have enabled the root window, you can no longer click on Windows NT desktop shortcuts. You still have access to the Windows NT *Start* menu and the Task Bar.

For instruction on enabling the root window, see the file **\READ_X11.TXT** on the INTERIX CD-ROM.

The MWM Window Manager

The MWM Window Manager gives you the OSF/Motif 1.2.4 window controls and desktop environment, including icons.

To start *mwm*, run the command in the background. You may want to set it up as a desktop shortcut, as described in chapter 3.

Each user should also have an *mwm* startup file, **.mwmrc**, in his or her home directory. There is a sample resource file on your system as **\$ _ROOT/etc/mwmrc.usr**. It must be customized before it can be used.

To display the Motif root window menu, hold down the left mouse button on any empty section of the root window. To display the sessions menu, hold down the right mouse button on any empty section of the root window.

If you're running the X11 Server in multiple-window mode, you must enable the root window before you can display the Motif root window or sessions menus. See the instructions in the file **READ_X11.TXT** on the INTERIX CD-ROM.

Further Reading

A comprehensive description of the X Window System is beyond this book. For more information about X or Motif, check:

X Window System User's Guide, by Valerie Quercia and Tim O'Reilly, Volume 3 of the X Window Series, published by O'Reilly & Associates, Inc.

The X Window Series books are the definitive guides.

INTRODUCTION TO REGULAR EXPRESSIONS

BASIC REGULAR EXPRESSIONS

EXTENDED REGULAR EXPRESSIONS

CHARACTER SETS

EXAMPLES

CHAPTER 9

POSIX Regular Expressions

This chapter is a brief guide to POSIX regular expressions. If you are familiar with regular expressions, you may want to read this chapter anyway. Regular expressions as defined by POSIX are slightly different from traditional regular expressions.

Introduction to Regular Expressions

A regular expression is a shorthand description for a pattern of characters. Regular expressions are used for text searching and for text search-and-replace operations, both of which require finding a string of characters on a line. Regular expressions are used in the utilities *awk*, *ed*, *ex*, *expr*, *egrep*, *grep*, *more*, *sed*, and *vi*.

Regular expressions do have some limitations:

- Regular expressions don't normally span lines (though you can search across line breaks with *grep*).
- Some patterns are very difficult to express as regular expressions.

When talking about regular expressions, we talk about the *pattern* and about the characters that the pattern *matches*. This is similar (but not the same) to the shell's wildcards: a file specifier of "*" matches all file names, a shell wildcard specifier of "A*" matches all filenames that

start with A, and so on. Regular expressions do much the same thing, but they are much more powerful. They're also more complicated.

Most text characters stand for themselves. The letter 'a' (for example) in the pattern matches letter 'a' in the string being searched. A simple text string like "hello" matches "hello"—it is said to match itself.

Some text characters in the pattern can stand for more than a character or characters other than themselves; these are called *metacharacters*. For example, a dot (.) matches any single character (like the ? wildcard in the shell).

A pattern is built of smaller patterns. For example, the pattern "ab" is made of two patterns, "a" and "b", and it matches "ab" — it finds the string "ab" in the line. The pattern "a." is made of the patterns "a" and "." and it matches "a" followed by any other character...but it *won't* match an "a" at the end of a line (because there's no character following the "a").

Some of the metacharacters change how other patterns are interpreted. For instance, * has a special meaning only if it comes after another pattern; otherwise, it just matches "*".

Utilities are allowed to add their own metacharacters; for example, *ex* and *vi* have several metacharacters found nowhere else. (See the *vi* reference page.)

There are two different kinds of regular expressions, *basic* regular expressions and *extended* regular expressions. They use different metacharacters. As the names imply, extended regular expressions are more flexible and powerful than basic regular expressions.

Both kinds of regular expressions make use of a construct called a *character set* or a *bracket expression*; see "Character Sets" for a detailed description.

Basic Regular Expressions

Basic regular expressions are used in the utilities *ed*, *ex*, *expr*, *grep*, *more*, *sed*, and *vi*. They use the following metacharacters:

* . ^ \$ [] \ \ (\) \ { \ }

The chart summarizes how they are used.

Pattern	Matches
.	Matches any single character.
^	Matches the beginning of a line, but only if it occurs at the beginning of the regular expression.
\$	Matches the end of a line, but only if it occurs at the end of the regular expression.
[c]	A character set; it matches any one of the characters between the brackets — “[abc]” matches “a” or “b” or “c”. See “Character Sets” for more information.
[^c]	Matches any one character that is <i>not</i> between the brackets — “[^abc]” matches any character <i>except</i> “a” or “b” or “c”. See “Character Sets” for more information.
<i>pat</i> *	Changes the preceding pattern <i>pat</i> to mean “zero or more occurrences of that pattern” — the pattern “a” matches “a” but “a*” matches “”, “a”, “aa”, “aaa” and so on.
\c	Matches the literal character <i>c</i> , except where noted below. The digits 1 through 9 have a special meaning for <i>c</i> ; see “\n”, below.
\(<i>pat</i> \)	Group the characters between the parentheses as a single pattern. For example, “ban*” matches “ba”, “ban”, “bann” and so on, but “b\(<i>an</i> \)” matches “b”, “ban”, “banan” and so on. This is also used for substitution; see “\n”, below.
<i>pat</i> \{ <i>n</i> \}	Match <i>n</i> occurrences of the pattern <i>pat</i> .
<i>pat</i> \{ <i>n</i> ,\}	Match <i>n</i> or more occurrences of the pattern <i>pat</i> .
<i>pat</i> \{ <i>n</i> , <i>l</i> \}	Match between <i>n</i> and <i>l</i> occurrences of the pattern <i>pat</i> ; <i>n</i> must be less than <i>l</i> .
\n	Match the <i>n</i> ’th grouping in the pattern, grouped with \(\ \). Given the pattern \(\a\)\(b\)\(c\), \1 stands for “a”, \2 for “b”, and \3 for “c”. <i>N</i> must be a digit from 1 to 9.

Extended Regular Expressions

Extended regular expressions are used in the *awk* and *egrep* utilities. They change the meanings of some of the metacharacters and add others:

* . ^ \$ [] \ ? + | () { }

- The first seven metacharacters (*, ., ^, \$, [,], \) keep the same meaning as for basic regular expressions.
- Bracket expressions are the same as for basic regular expressions.
- Instead of \ (and \), extended regular expressions use (and); instead of \ { and \}, extended regular expressions use { and }. To match (, use \ (; to match), use \); to match {, use \ {; and to match }, use \ }.

- The metacharacters ?, + and | are new, and have these meanings:

pat? Match 0 or 1 occurrences of the pattern *pat*.

pat+ Match 1 or more occurrences of the pattern *pat*.

pat1 | *pat2* Match either *pat1* or *pat2*. The pattern “hot dog | hamburger” matches either “hot dog” or “hamburger”. The pattern “[Ss]tor(y | ies)” matches “Story”, “story”, “Stories” and “stories”.

Character Sets

A character set (or a “bracket expression”) is a set of characters to match. Character sets are placed in brackets, like this, [*expr*], which is where the name bracket expression comes from. A character set usually matches any one of the characters described by the expression *expr*. Some metacharacters have different meanings inside a bracket expression.

Character sets are useful and frequently misunderstood. POSIX extends them to include *collating sequences* and *equivalence classes* and provides some predefined character sets.

Collating sequences and equivalence classes were introduced to handle 8-bit and multibyte locales; users in the C and POSIX locales have relatively little use for them. “Collation” is the ordering of a character set; for example, the characters “a–z” are guaranteed to sort with “a” before “b” (and so on), no matter what representation is used within

the computer. Collation is performed based on collating elements; a collating element can be a single character, a multi-character sequence that collates as if it were a single character, or a collating-sequence name.

An equivalence class defines a set of characters with the same value for the purposes of collation. For example, if the locale supports accented characters, all accented “a” characters might be given the same weight for collation so à, á, â and ã sort identically.

Here are the characters that can be put inside a bracket-expression and their meanings:

Pattern	Matches
<i>c</i>	Any character represents itself. <code>[abc]</code> matches “a” or “b” or “c”. You can also represent a range of characters with a “-”: <code>[a-c]</code> is identical to <code>[abc]</code> . To include “-” in the set, make it the first or last character. To include “]” in the set, make it the first character; the character set to find “[, “]” or “-” is <code>[][-]</code> .
<code>* \ . [</code>	Match themselves; they lose their special meanings when inside bracket expressions.
<code>^</code>	If this is the first character in a bracket-expression, it inverts the meaning of the expression — the expression matches any one character that is <i>not</i> between the brackets.
<code>[.c.]</code>	Represents a collating element. <i>c</i> can be one or more characters, so long as it represents a single collating element. In the C and POSIX locales, all characters are single bytes: The regular expression <code>[[.a.]-[.z.]]</code> is identical to <code>[a-z]</code> .
<code>[=c=]</code>	Represents an equivalence class. The one or more characters represented by <i>c</i> are all treated as the same for collating purposes.
<code>[:classname:]</code>	Represents a character class. For example, the class <code>[:punct:]</code> represents all punctuation characters.

The following character classes are defined:

<code>[:alnum:]</code>	Matches any alphabetic character or a digit from 0 to 9.
<code>[:alpha:]</code>	Matches any alphabetic character.
<code>[:blank:]</code>	Matches the space and tab characters.
<code>[:cntrl:]</code>	Matches any control character.
<code>[:digit:]</code>	Matches any decimal digit, from 0 to 9.
<code>[:graph:]</code>	Matches any printing character <i>except</i> space.
<code>[:lower:]</code>	Matches any lowercase letter.
<code>[:print:]</code>	Matches any printable character except for the space character.
<code>[:punct:]</code>	Matches any punctuation character: any printing character that isn't a letter or a digit.
<code>[:space:]</code>	Matches any of the white-space characters space, form-feed, horizontal tab, vertical tab, and carriage return.
<code>[:upper:]</code>	Matches any uppercase letter.
<code>[:xdigit:]</code>	Matches any letter or digit that could be part of a hexadecimal number: the digits 0 to 9, the letters A through F, and the letters a through f.

Examples

Here are some examples. A space character is represented by `␣`.

Basic Regular Expressions

Match	Expression
Empty line	<code>^\$</code>
One or more spaces	<code>␣*</code>
Any character except <code>[</code> , <code>]</code> , or <code>-</code>	<code>[^[-]]</code>
Any uppercase letter	<code>[A-Z]</code> or <code>[:upper:]</code>
Local North American telephone number	<code>[0-9]\{3\}-[0-9]\{4\}</code>
Three or more uppercase letters	<code>[A-Z]\{3,\}</code>
Two or three occurrences of the word "and", possibly separated by spaces	<code>\(and␣*\)\{2,3\}</code>
Swap two words in <i>ed</i> , <i>ex</i> , or <i>vi</i>	<code>s/\([^\␣][^\␣]*\)␣\([^\␣][^\␣]*\)\/\2\1</code>
A valid C-language variable name	<code>[_A-z][_A-z0-9]*</code>

Extended Regular Expressions

Match

North American telephone number

Zero or one spaces

One or more spaces

“March” or “April”

“March”, “Mar”, “April” or “Apr”

“Story”, “story”, “Stories” or “stories”

Expression

[0-9]{3}-[0-9]{4}

[]?

[]+

March | April

(Mar(ch)?) | (Apr(il)?)

[Ss]tor(y | ies)

CHAPTER 10

Glossary

This section provides definitions of terms used in the documentation. Some of these are POSIX.1 terms, others are unique to the INTERIX environment.

Background process

A background process is a member of a background process group; it is normally (though not always) denied permission to read or write to the controlling terminal. Background processes do not receive character special signals such as INTR (Control-C) or SUSP (Control-Z). The shells allow you to create background processes by using the *bg* command or by terminating the command line with **&**.

Controlling Process

A session leader with a controlling terminal is a controlling process.

Controlling Terminal

A terminal that is associated with a session is known as the controlling terminal for that session and its members.

Current Working Directory

For resolving pathname searches, each process has a current working directory associated with it. All pathnames that are not

rooted (that do not begin with /) are prefixed by the current working directory. For example, if the current working directory is **//C/ Interix/etc** the filename **motd** is assumed to refer to the file **//C/Interix/etc/motd**.

Curses

A package of library routines used in programs, curses provides a standard character graphics interface for terminal display. Curses programs usually require the user to have the *TERM* environment variable set.

Daemon

A background process that intentionally doesn't terminate. Daemons are frequently used as server processes.

Descriptor

An integer assigned by the system when a file is referenced by *open()* or *dup()*, which uniquely identifies an access path to that file from a given process or any of its children.

Directory

A directory is a special type of file that contains entries that are references to other files. Directory entries are called links. A directory contains at least two links, *.* and *..*, referred to as "dot" and "dot-dot" respectively. Dot refers to the directory itself and dot-dot refers to its parent directory. (In the INTERIX Software Development Kit, the *readdir()* function does not return entries for dot or dot-dot.)

Domain

The term "domain" is used in both the Windows NT network model and in the Internet addressing model; the meanings are not interchangeable.

A Windows NT domain is a collection of computers which share a central security database; it's roughly equivalent to a single centralized password database, such as Network Information Services. Since each Windows NT machine has its own security database, every Windows NT belongs to *at least* one domain. Users in different domains are entirely separate; they need not be the same person, whether or not they have the same login name. The domain is a part of the user's security authentication, along with the user ID.

An Internet domain is a node in the hierarchical tree structure of internet addresses. For example, “com” is a top-level domain usually used to identify commercial organizations.

Effective User ID, Effective Group ID, and Group Access List

Access to system resources is governed by two values: the effective user ID and the group access list. In the INTERIX subsystem, the effective user ID of a process is the same as its real user ID. See the discussion of “appropriate privileges” in chapter 3.

Before permitting a process access to resources, the system checks to see that the group ID of the process is on the list of permitted groups—the group access list. (In POSIX.1, the group access list is known as the set of “supplementary group IDs”.) The group access list is used only in determining resource accessibility. Access checks are performed as described below in “File Access Permissions”.

The first member of the group access list is also known as the effective group ID. It is a member of the list of supplementary group IDs.

FIFO

A special type of file; data read from a FIFO is read in the order it was written, first-in, first-out.

File

Any object in the system that can be written to, or read from, or both. Files have access permissions, which determine whether a process can perform an operation on the file, and a type. The INTERIX subsystem supports the following types of files: normal, directory, FIFO, special block and special character.

File Access Permissions

Every file in the file system has a set of access permissions. These permissions are used in determining whether a process may perform a requested operation on the file (such as opening a file for writing). The following discussion relates only to file access permissions in the INTERIX subsystem; see chapter 4 for a more detail discussion of the Windows NT security model and the interaction of Win32 permissions with POSIX file permissions.

Access permissions are established at the time a file is created. They may be changed at some later time through the *chmod()* call. File access is broken down according to whether a file may be: read, written, or executed. Directory files use the execute permission to control if the directory may be searched.

File access permissions are interpreted by the system as they apply to three different classes of users: the owner of the file, those users in the file's group, anyone else. Every file has an independent set of access permissions for each of these classes. When an access check is made, the system decides if permission should be granted by checking the access information applicable to the caller.

Read, write, and execute / search permissions on a file are granted to a process if:

- The process's effective user ID has the appropriate privileges. (Execute privilege will not let you execute a non-executable file.)
- The process's effective user ID matches the user ID of the owner of the file and the owner permissions allow the access.
- The process's effective user ID does not match the user ID of the owner of the file, and either the process's effective group ID matches the group ID of the file, or the group ID of the file is in the process's group access list, and the group permissions allow the access.
- Neither the effective user ID nor effective group ID and group access list of the process match the corresponding user ID and group ID of the file, but the permissions for "other users" allow access.

Otherwise, permission is denied.

Filename

A filename is a single component of a pathname and identifies an ordinary file, special file, or directory within a directory. A filename is always unique within its directory, but it doesn't need to be unique within the file system.

Names can be up to {NAME_MAX} characters long. On NTFS partitions, filename characters may be selected from the set of all ASCII characters excluding 0 (NUL) and the code for / (slash). The actual characters available may be limited by the filesystem. For example, the FAT and HPFS file systems are not case-sensitive. *Portable* filenames should be constructed from characters in the portable filename character set ([A-Za-z0-9][._-]) and should not start with a hyphen.

Avoid using *, ?, [or] as part of filenames because the shell attaches special meaning to these characters; manipulating files containing those character is always tedious and difficult.

File Serial Number

A number uniquely identifying a file in that file system. The file serial number roughly corresponds to the inode number.

Foreground Process

A process that's a member of a foreground process group is a foreground process. It has privileges that background process groups do not, notably the ability to read and write the controlling terminal.

Group

A set of users, identified by a group ID. Groups are typically used to collect users into related sets, such as projects or departments. Each user belongs to at least one group, and can belong to several.

Group ID

A non-negative integer that identifies a group of system users.

Link

A link is a directory entry. It is possible for a single object, a file, to have more than one directory entry and therefore more than one name. To have a link, a file must be on a New Technology File System (NTFS) device, and the links cannot span devices.

OPENNT_ROOT

An environment variable whose value is the installation directory for the INTERIX file hierarchy, normally **//C/Interix**.

Orphaned Process Group

A process group in which the parent of every member is either a member of the process group, or is not a member of the group's session. Therefore, to *not* be orphaned, a process group must have a member whose parent is not in the same process group but is in the same session.

Parent Process ID

A new process is created by a currently active process; see the reference for the *fork()* system interface. The parent process ID of a process is initially the process ID of its creator. If the creating process exits, the parent process ID of each child is set to the special ID of 1.

Pathname

A pathname identifies a file or directory within a file system. It is a NUL-terminated character string starting with an optional slash /,

followed by zero or more directory names separated by slashes, optionally followed by a filename. The total length of a pathname must be less than {PATH_MAX} characters.

If a pathname begins with a slash, the path search begins at the root directory and the pathname is considered “rooted” or “absolute.” Otherwise, the search begins from the current working directory and the pathname is considered “relative.” A slash by itself names the root directory. An empty pathname refers to the current directory.

Pathnames are assumed to be on the same drive as the working directory. In the INTERIX subsystem, you can specify a drive by starting the absolute pathname with two slashes and an uppercase character; the uppercase character identifies the drive for the pathname. For example, a pathname beginning with **//D** refers to a file or directory on the D: drive.

Pipe

A pipe is a first-in, first-out queue that connects the two file descriptors given to the *pipe()* function. More traditionally, it is a means of connecting the standard output of one process to the standard input of another. Pipes on INTERIX behave identically to FIFO files, but they do not show up in the file hierarchy.

Process

A single thread of control within an address space, and its required system resources. A process can be thought of as a single program while it is executing. A process can create a new process using the *fork()* call. The process that uses *fork()* is known as the parent process, and the new process is the child process.

Process Group

Each active process is a member of a process group; all processes in a process group can be signaled together. All processes in a process group are identified by a non-negative integer called the process group ID, which is the process ID of the group leader.

Process ID

Each active process in the system is uniquely identified by a non-negative integer called a process ID. Process IDs are re-used.

Real User ID and Real Group ID

Each user on the system is identified by a positive integer termed the real user ID.

Each user is also a member of one or more groups. One of these groups is distinguished from others and used in implementing accounting facilities. The positive integer corresponding to this distinguished group is termed the real group ID.

All processes have a real user ID and real group ID. These are initialized from the equivalent attributes of the process that created it. The shell has the user and group IDs of the user who invoked it.

Root Directory

Each process has associated with it a concept of a root directory for the purpose of resolving pathname searches. See also “Current Working Directory”.

Root User

Traditionally, the system administration account (or the “super-user”) is named “root”, has user ID 0 and is given special privileges. There is no root user under INTERIX.

Screen

In the X Window System, a screen is a physical display unit.

In the curses library, a screen is a curses window that is as large as the terminal screen (it starts at the upper left corner and encompasses the lower right corner). One screen, *stdscr*, is automatically provided for the programmer.

Server

A process that waits for a client to contact it, and that then provides a service. Daemons often act as servers (such as the telnet daemon, the FTP daemon, the HTTP daemon). The X server provides a service (bitmapped screen display) to an X client.

Session

A session is a set of one or more process groups. A session is created by a successful call to *setsid()*, which causes the caller to become the only member of the only process group in the new session. This process becomes the session leader.

Session Leader

A process that has created a new session by a successful call to *setsid()*, is known as a session leader. Only a session leader may acquire a terminal as its controlling terminal; other processes inherit the controlling terminal from their parent process.

Shell

A command language interpreter that executes programs on behalf of the user; the user's environment and protection from the operating system. INTERIX provides two shells: the KornShell (*ksh*), and the C shell (*tcsh*).

Special Process ID

When a process becomes an *orphaned process*—that is, when its parent process ends—its parent process ID is assigned to process 1.

Terminal

A input-output device that is recognized by the system as a character special device file. Normally the terminal becomes the standard input, output and error of an application.

Terminal Screen

The screen of the terminal. In the curses library, this term is used to mean the software package's idea of what the terminal display currently looks like (what the user sees now).

Terminal Process Group ID

A terminal may be acquired by a session leader as its controlling terminal. Once a terminal is associated with a session, any of the process groups within the session may be placed into the foreground by setting the terminal process group ID to the ID of the process group. This facility is used to arbitrate between multiple jobs contending for the same terminal.

User ID

A non-negative integer that identifies one user of the system.

Window

In the curses library, the window is a data structure describing a subrectangle of the screen, possibly the entire screen. You can write to a window as though it were a miniature screen, scrolling independently of other windows on the physical screen.

Index

Symbols

- `/dev/console` 63
- `/dev/null` 51
- `/dev/proc` 51
- `/dev/ptmx` 51
- `/etc/profile.usr` 14
- `#!` notation
 - for Win32 programs 39
- `$psxjunk` 30
- `$HOME/.profile` 14
- `$OPENNT_ROOT/etc/mwmrc.usr` file 73
- `$OPENNT_ROOT/etc/syslog.conf` file 62
 - examples 64
- `$OPENNT_ROOT/usr/share/termcap` file 43, 44
- `$OPENNT_ROOT/var/adm/log/logger` 61
- `$OPENNT_ROOT/bin/csh` 90
- `$OPENNT_ROOT/bin/ksh` 90
- `$OPENNT_ROOT/bin/sh` 90
- `$OPENNT_ROOT/bin/tcsh` 90
- `$OPENNT_ROOT/usr/share/man` 67
- `.mwmrc` file 73
- `.profile`
 - creating. *See* `$HOME/.profile`
- `/dev/null` 44
- `/dev/tty` 44, 51
- `/etc/csh.login` 13
- `/etc/profile` 14
- `/etc/profile.lcl` 14
- `^M` characters 31

A

- access control entry. *See* ACE
- access control list. *See* ACL
- ACE 27
 - CACLS utility 40
- ACL 27
- `apropos` command 68
- `at` utility 58
 - removing a command 59
- AUX filename 41

B

- background process
 - definition 83
- batch utility 59
- `bg` command 83
- bracket expression 76
- Bypass Traverse Checking right 29

C

- C shell 90
- CACLS utility 40
- `cat32` utility 40
- character classes 80
- character set 76
- `chmod()` 85
- command pipelines
 - with Win32 apps 40
- Commands and Utilities
 - default location 11
- COMx filename 41
- console window 43, 45
- control characters
 - terminal (`stty`) 43
- controlling process 83
- controlling terminal 83
- `copyrite.pdf` 66
- `cpio` utility
 - tape drives 52
- cron service 55
- `cronadmin` utility 55
- `crontab` utility 55, 56
- `cuintro.pdf` 66
- `curef.pdf` 66
- current working directory 83
- curses
 - definition 84
- `curses.pdf` 66
- customer support 9
- `cut-and-paste` 45
 - buttons 45

D

- daemon
 - definition 84
- daemons 55–64
 - inetd 60
 - rlogind 60
- dd utility 53
 - tape drives 52
- descriptor 84
- devices
 - list of virtual 51
- df command 30
- directories
 - \$psxjunk 24, 30
 - . 84
 - .. 84
 - execute permission 85
 - root 88
- directory definition 84
- directory permissions
 - and chdir() 29
- domain
 - definition 84
- domains
 - workgroup servers without 61
- dup() 84

E

- effective group ID 85
- effective user ID 85
- elvis utility 66
- end-of-line
 - MS-DOS vs POSIX 31
- environment variables
 - and services 56
 - DISPLAY 71
 - HOME 13
 - LOGNAME 13
 - MANPATH 67
 - OPENNT_ROOT 13
 - PATH 13, 14
 - SYSTEMROOT 37
 - TERM 13, 16, 44, 49
 - TMPDIR 13
 - TZ 56
 - USERNAME 13
 - XAPPLRESDIR 72

- event logs (syslogd) 61
- Event Viewer 61

F

- FIFO 85
- file definition 85
- file descriptor 84
- file system
 - hard links 30
- filenames 86
 - reserved Win32 41
- files
 - \$OPENNT_ROOT/etc/mwmrc usr 73
 - \$OPENNT_ROOT/usr/share/termcap 43
 - \$OPENNT_ROOT/var/adm/log/logger 61
 - \$OPENNT_ROOT/etc/profile 14
 - \$OPENNT_ROOT/etc/syslog.conf 62
 - .exrc 14
 - //C/Interix/usr/share/termcap 43
 - /dev/null 44
 - /dev/tty 44
 - /etc/csh.login 13
 - /etc/profile 14
 - /etc/profile.lcl 14
 - /etc/profile usr 14
 - access permissions 85–86
 - owned by "Everyone" 30
 - READ_X11.TXT 71, 72
 - reserved names under Win32 41
 - serial number 87
 - startup 14
 - text, converting MS-DOS to Interix 15
 - types 85
- flip utility 15
- fork() 87
- Free Software Foundation 66
- FTP access, Softway Systems 9
- functions
 - chmod() 85
 - dup() 84
 - fork() 87
 - open() 84
 - readdir() 84
 - setsid() 89

G

- getstart.pdf 65

GLPL.TXT 66
GPL.TXT 66
group 87
group access list 85
group “Everyone” 30
group ID 87

H

hard links 30
home directory 40
 ownership 40
 setting 14
HOME environment variable 13

I

inetd service 60
inode 87

K

kill utility
 killing Win32 processes 39
killing processes 39
KornShell 49, 90
 aliases 35–36
 cmd_intitle option 49
 displaying the current command in title bar 49
ksh. *See* KornShell

L

line endings
 CR-LF vs. LF 31
link 87
login shell 12
LoginDefaultDomain registry entry 61
LOGNAME environment variable 13
LPT1 filename 41

M

man utility 67–68
 section numbers 67
MANPATH environment variable 67
manual sections 67

miscinfo.pdf 66
Motif root window menu 73
Motif sessions menu 73
mt utility 23, 52
mwm
 startup file 73
mwm window manager 72

N

NAME_MAX 86
NTconsole terminal type 43
NTFS 30, 86, 87
NUL filename 41
nvi utility 66

O

open() 84
OPENNT_ROOT environment variable 13
orphaned process group 87

P

PATH 14
PATH environment variable 13
 adding Win32 commands to 35
 default 14
PATH_MAX 88
pathnames 87
 \$OPENNT_ROOT/etc/profile 14
 \$OPENNT_ROOT/etc/profile.lcl 14
 \$OPENNT_ROOT/usr/share/man 67
C: 11
 device identifier 30
 drive specifiers 88
 POSIX format 11
 relative 88
 rooted 88
 separator 30
pax utility
 tape drives 52
pcd1.pdf 66
pcd2.pdf 66
permissions
 Special Access permission. *See* ACE
pipe 88
POSIX 7
 conformance documents 66

- regular expressions 75
- POSIX file system 31
- posixpath2nt utility 39
- PRN filename 41
- process
 - definition 88
- process group 88
- process group ID 88
 - terminal 90
- process ID 88
 - parent 87
 - special 90
- ps utility 51
 - listing Win32 processes 39
- pseudo terminals 23, 51
- psxrun.exe utility 55

R

- READ_X11.TXT file 71
- readdir() 84
- real group ID 88–89
- real user ID 88–89
- registry entries 61
- regular expression 80
 - equivalence classes 78
- regular expressions
 - basic 76
 - bracket expressions 78
 - character classes 80
 - collating sequences 78
 - description 75
 - equivalence classes 79
 - extended 78
- reserved filenames
 - under Win32 41
- rlogind daemon 60
- root directory 89
- root user 24. *See also* superuser
- rsh
 - remote shell utility 22
 - restricted shell utility 22

S

- screen
 - definition 89
- sdkintro.pdf 66
- server

- definition 89
- Service Control Manager 56
- service utility 55
- services 55–64
 - cron 55, 56–59
 - cronadmin 55
- session 89
- session leader 89
- setsid() 89
- setuid and setgid mode bits 24
- sh utility 12
- shell 90
 - \$OPENNT_ROOT/bin/ksh 90
 - \$OPENNT_ROOT/bin/sh 90
 - \$OPENNT_ROOT/bin/tcsh 90
 - startup files 14
 - user and group ID 89
 - wildcards 30
- shells
 - \$OPENNT_ROOT/bin/csh 90
- showcmd shell variable (C shell) 49
- siref.pdf 66
- Softway Systems
 - addresses 9
 - communicating with 9
- startup files 14–15
- stty
 - control characters 43
- supplementary group ID list 85
- syslogd
 - starting 62
- SYSTEMROOT environment variable 37

T

- tape devices
 - blocking factors 52
- Tape Drives 52–54
- tape drives 23
 - controlling with mt 52
- tar utility
 - tape drives 52
 - tape portability 52
- Task Manager 39
 - killing Interix processes 39
- tcsh utility 12
- technical notes 69
- technical support 9
- telnet

- and on-line documentation 66
- and pseudo terminals 51
- telnet utility
 - default login domain 61
 - default shell 61
- TERM environment variable 13, 16, 44, 49
- termcap 43
- termcap file 43
- terminal 90
- terminal screen
 - definition 90
- terminal type
 - ansi 44
 - opennt 13, 44
- terminal types
 - changing 49
 - opennt 49
 - opennt-50 49
 - opennt-60 49
 - opennt-w 49
 - tall 49
 - wide displays 49
- text files
 - format 31
- TKILL.EXE 39
- TMPDIR environment variable 13
- twm utility 72
- twm window manager 72
- TZ environment variable 56

U

- universal naming convention 24
- user and group names 30
- user “Everyone” 30
- user id 90
- User Manager 14, 40
- user rights on Windows NT 29
- USERNAME environment variable 13
- utilities
 - at 58
 - batch 59
 - bg 83
 - CACLS 40
 - cat32 40
 - cron 55
 - cronadmin 55
 - crontab 55, 56
 - dd 53

- df 30
- elvis 66
- flip 15
- kill 39
- man 67–68
- mt 23, 52
- nvi 66
- posixpath2nt 39
- ps 39, 51
- rsh (remote shell) 22
- rsh (restricted shell) 22
- service 55
- tar 52
- twm 72
- xcalc 71
- xlsfonts 71
- xrdb 71
- xterm 23, 48, 71

V

- virtual devices 51–54

W

- Win32 applications 30
- window
 - definition 90
- window managers 72
- Windows NT Resource Kit 40
- Windows NT Resource Kit, Volume 1 40
- workgroup servers 61

X

- X11 71–73
 - enabling root window 72
 - root window menus 72
 - window managers 72
- X11 Server
 - enabling root window 73
- X11 server
 - information about 71
- x11 server
 - window display modes 72
- x11.pdf 66
- X11R5
 - window managers 72
- XAPPLRESDIR environment variable 72
- xcalc utility 71
- xlsfonts utility 71
- xrdb utility 71
- xterm

