UNIX* System V/386 Release 3.2
SDS Release Notes

INTERACTIVE

product family

UNIX® System V/386 Release 3.2

Software Development Set Release Notes

Note that some of the manual pages in Appendix A are superseded by those in the *INTERACTIVE Software Development System Guide and Programmer's Reference Manual*.

NOTICE

The information in this document is subject to change without notice. AT&T assumes no responsibility for any errors that may appear in this document.

CONTENTS

UNIX SYSTEM V/386 RELEASE 3.2 SOFTWARE DEVELOPMENT SET RELEASE NOTES

Introduction

Overview

These *Release Notes* contain information about the Software Development Set (SDS) package. The SDS package is useful to programmers who:

- Want to develop C language programs
- Do extensive programming in the C language
- Want to enhance the efficiency of a C program written in a UNIX system environment
- Need tools to do advanced programming and symbolic debugging
- Want to work with shared libraries
- Work in an environment where it is necessary to track and maintain versions of files and programs
- Want to optimize and streamline development of interactive, character-oriented, C application programs.

The Software Development Set runs on a computer running AT&T 386 UNIX System V/386 Release 3.2.

The SDS software package is made up of two parts as follows:

- C Software Development Set (CSDS)
- Extended Terminal Interface (ETI).

Conventions Used in This Document

In these *Release Notes*, certain typesetting conventions are followed when command names, command line format, files, and directory names are described. There are also conventions for displays of terminal input and output.

- You must type words that are in **bold** font exactly as they appear. Also, commands, filenames, and directory names appear in **bold**.
- Words in *italics* are variables; you substitute the appropriate values. These values may be filenames or they may be data values.
- CRT or terminal output and examples of source code are presented in `constant-width` font.
- In output and source code examples, a backslash (\) at the end of a line indicates that the line wraps around without a break.
- A command name followed by a number, for example, **prof**(1), refers you to that command's manual page, where the number refers to the section of the manual. These manual pages appear in the *AT&T UNIX System V/386 Release 3.2 Programmer's Reference Manual* unless otherwise noted.

Contents of the Release

The Software Development Set (SDS) comes in one set of five diskettes (four diskettes for CSDS and one diskette for ETI), the contents of which are displayed in the following table.

Table 1: SDS Utilities

Directory	Files		
/bin (CSDS)	ar as cc chkshlib conv convert	cprs dis dump gencc ld list	lorder make mkshlib nm size strip
/etc (CSDS)	install		
/lib (CSDS)	basicblk cm4defs comp cpp crt0.o crt1.o	crtn.o libc.a libc_s.a libld.a libm.a libPW.a	libx.a mcrt0.o mcrt1.o optim pcrt1.o pcrt0.o
/usr/add-on/include (ETI)	chartam.h form.h kcodes.h menu.h message.h	pbf.h print.h subcurses.h tam.h tamwin.h	temp.h track.h wind.h
/usr/add-on/include/sys (ETI)	font.h iohw.h	mouse.h signal.h	window.h

Table 1: SDS Utilities (Continued)

Directory	Files		
/usr/bin	admin	delta	sact
	cb	get	sccsdiff
	cdc	lex	sdb
	cflow	lint	tsort
	comb	lprof	unget
(CSDS)	cscope	m4	val
	ctc	prof	vc
	ctcr	prs	what
	ctrace	regcmp	yacc
	cxref	rmdel	
/usr/bin (ETI)	captoinfo	infocmp	tic
	tput		

Table 1: SDS Utilities (Continued)

Directory	Files		
/usr/include (CSDS)	a.out.h	malloc.h	sgtty.h
	aouthdr.h	math.h	signal.h
	ar.h	memory.h	stand.h
	assert.h	mnttab.h	stdio.h
	core.h	mon.h	storclass.h
	ctype.h	nan.h	string.h
	dial.h	nlist.h	stropts.h
	dirent.h	nsaddr.h	strselect.h
	errno.h	nserve.h	syms.h
	fatal.h	poll.h	sys.s
	fcntl.h	prof.h	termio.h
	filehdr.h	pwd.h	time.h
	ftw.h	regexp.h	tp_defs.h
	grp.h	reloc.h	ttysrv.h
	ieeefp.h	rje.h	unistd.h
	ldfcn.h	scnhdr.h	ustat.h
	limits.h	sd.h	utmp.h
	linenum.h	search.h	values.h
	macros.h	setjmp.h	varargs.h
/usr/include (ETI)	curses.h	menu.h	tiuser.h
	eti.h	panel.h	unctrl.h
	form.h	term.h	

Table 1: SDS Utilities (Continued)

Directory	Files		
/usr/lib (CSDS)	basicblk dag flip libcrypt.a libg.a libl.a libmalloc.a libprof.a	liby.a lint1 lint2 llib-lc llib-lc.ln llib-lm llib-lm.ln llib-lmalloc.1	llib-port llib-port.ln lpfx nmf xcpp xpass yaccpar
/usr/lib (ETI)	libcrypt.a libform.a libmenu.a libpanel.a libtam.a libtermcap.a libtermlib.a	llib-lcurses llib-lcurses.a llib-lcurses.ln llib-lform llib-lform.ln llib-lmenu	llib-lmenu.ln llib-lpanel llib-lpanel.ln llib-ltam llib-ltam.ln tamhelp
/usr/lib/ctrace (CSDS)	runtime.c		
/usr/lib/help (CSDS)	ad bd cb cm cmds	co de default ge he	prs rc un ut vc
/usr/lib/help/lib	help	help2	

Table 1: SDS Utilities (Continued)

Directory	Files		
/usr/lib/lex (CSDS)	ncform	nrform	
/usr/lib/libp (CSDS)	libc.a libx.a	libm.a	libmalloc.a
/usr/lib/tabset (ETI)	3101 beehive	std teleray	vt100 xerox1720
/usr/options (CSDS)	csoftw.name		
/usr/options (ETI)	graphi.name	terminf.name	
/usr/src/lib/eti/demo (ETI)	form0.c form1.c	form2.c menu0.c	menu1.c

Software Overview

The SDS package has two major parts: the C software development set (CSDS) and the extended terminal interface (ETI). CSDS can be used for developing, debugging, and improving the efficiency of C language programs. ETI is a set of libraries that promotes fast development of screen management applications. These two parts of the SDS package are discussed in the following subsections.

CSDS

CSDS is a collection of tools and utilities that aid you in:

- Developing C language programs
- Advanced programming, symbolic debugging, and improving C language program efficiency.
- Keeping a history of source code files by recording changes made to these files along with comments on each version.

The C Programming Language Development Tools

The main C programming language development tool is the compiler, and is called by the command **cc**. The other programming development tools discussed in this section are the C preprocessor, optimizer, assembler, link editor, tools for manipulating object files, and libraries.

C Compiler

The C compiler supports the C language as specified in *The C Programming Language*. The significant extensions to the language include the following:

- Arbitrary length names for variables and function names
- Structure assignments and arguments
- Functions returning structure values
- Enumerated data types
- Multiple external variable declarations

- Assembly language escapes from C
- Insertion of arbitrary strings into object modules (useful for version control)
- Floating point support in conformance with the *Standard for Binary Floating-Point Arithmetic*, ANSI/IEEE Std 754-1985
- Data type **void**
- Additional preprocessor directives.

cc Command

The **cc** command, the major command of CSDS, calls the C compiler. The **cc** command also controls the other phases of compilation, and, unless programmers use options to specify otherwise, **cc** automatically calls the C preprocessor, assembler, and link editor phases. The command options have many uses, such as suppressing the assembler or link editor or invoking the optimizer. The **cc** command also passes some options to these other programs.

The **cc** command accepts files containing C source code as input. The result of the compilation process is an executable module named **a.out** that reflects the contents of the source files and any referenced library routines. The **cc** command also accepts source files that contain assembly language code as input and passes these files directly to the assembler.

C Preprocessor

The C preprocessor [**cpp**(1)] is automatically called whenever the **cc** command is given C source input. The preprocessor performs file inclusion and macro substitution.

Optimizer

The optimizer, an optional component in the compilation process, improves the efficiency of compiler-generated assembly language code. The optimizer reduces the space requirements and speeds the execution time of the resulting object code.

Assembler and Assembly Language

The assembler [**as**(1)] is available for developing applications that require close interaction with hardware, such as those needed to handle input/output devices and interrupts. The assembler converts assembly language code into a relocatable object module composed of machine code and symbolic information. This component provides assembly language programmers access to predefined macros using the UNIX operating system **m4** macro processor.

Link Editor

The link editor [**ld**(1)] combines relocatable object modules and libraries to produce either an absolute, executable load module or a relocatable object file for use in further link edits. Executable load modules are in the Common Object File Format (COFF). The link editor performs relocation, resolves external references, and incorporates symbolic debugging information into its output file. It searches libraries to resolve all external references and only loads library routines that define an unresolved external reference.

Tools for Manipulating Object Files

CSDS provides a variety of commands used to read and manipulate object files. Here is a list of some utilities with brief descriptions of their use:

ar	Groups files into a single, portable archive file commonly used as a library
cprs	Compresses object files by removing duplicate structure and union symbolic information
dis	Disassembles object files to allow assembly level debugging
dump	Prints selected parts of the named object files
lorder	Generates an ordered listing of object files for efficient library link editing
nm	Prints the symbolic information in an object file
size	Reports the number of bytes of text, initialized data, and uninitialized data (and their sum) included in an object module
strip	Reduces file storage overhead by removing symbolic information from an object file.

Libraries

CSDS comes with libraries for object files, access to system calls, input/output, string manipulation, mathematical functions, and memory allocation.

Advanced Programming Tools and Utilities

The CSDS package contains an extensive set of tools useful for advanced application programming, debugging, improving the efficiency of your programs, and aiding you in keeping track of the different versions of your programs.

Programming and Debugging Utilities

The programming utilities are specialized utilities helpful in the design and development of application programs and systems. The following list gives a short description of the major programming utilities.

cxref is a C cross-reference listing generator.

ctrace is a statement-by-statement execution trace facility.

cflow produces a graph of program dependencies.

lint detects faulty and non-portable code.

cb is a C code beautifier

regcmp compiles regular expressions

mkshlib(1) makes a shared library. Shared libraries is a feature of UNIX System V Release 3.0, and beyond, that allow several **a.out** files to simultaneously use the same object code.

chkshlib(1) checks a shared library.

sdb(1) a symbolic debugger used to examine C language executable files and core files and to provide a controlled environment for their execution. When testing C language programs symbolically, breakpoints can be set at executable lines of the source code. These breakpoints force the program to pause at the specified point so that an inspection can be made of the current state of the program.

make(1) a tool that helps you build and maintain up-to-date versions of programs. **make** simplifies the job of keeping track of which files depend on other files, recently modified files, files that need recompiling after changes, and the sequence of operations needed to make a new version of a program.

lex(1) a tool that generates programs to be used in simple lexical analysis of text. The **lex** tool reads a file containing specifications of strings to be matched and associated C code. Whenever the lexical analyzer produced by **lex** matches a specified string in its input, it executes the associated C code.

yacc(1) a tool (Yet Another Compiler-Compiler) that accepts both an LALR(1) grammar specification and associated C code fragments that represent actions to be taken when a found

grammar rule is reduced, and then produces a parser.

All of these utilities are described in the *AT&T UNIX System V/386 Release 3.2 Programmer's Guide* and the *UNIX System V/386 Release 3.2 Programmer's Reference Manual.*

Productivity Utilities

The CSDS package has three utilities that can help an experienced programmer enhance the efficiency of a C program written in a UNIX operating system environment. These utilities are a browser called **cscope** and two profilers, **lprof** and **prof**.

A browser is an interactive program that helps you examine source files by searching for functions, function calls, macros, and variables that you specify. When it finds them, the browser puts you into an editor at the specified location. Thus, instead of thumbing through a stack of printouts to learn code or locate a bug, you can specify a function or text string and let the browser find it. Then you have the option of examining that portion of code or editing it. Whether you want to familiarize yourself with a program or edit a source file, a browser can help you accomplish your task without your reading the code line by line.

The browser in CSDS, designed for use with C code, is called **cscope**. Programmers responsible for writing programs or maintaining existing programs will be able to edit their source code more efficiently with **cscope**. It is especially helpful for a programmer working on someone else's code.

A profiler is a tool that performs dynamic analysis or analysis of a program at run time; it accomplishes this in two phases. First, the profiler collects data about the code while a program is being executed. Then it displays this data in a readily accessible format. The profiler **lprof** provides line-by-line frequency profiling, reporting how many times each line of source code is executed. To obtain a more representative sample of program performance, you can run a program profiled with **lprof** more than once and then merge the data from the multiple runs. This information can be useful in every stage of software development: designing, prototyping, coding, testing, debugging, and maintenance.

The profiler **lprof** can also be used to determine which lines of source code are executed and how much of the code is exercised. These types of output can be obtained by using the **-x** option and the **-s** option, respectively. These options are convenient for programmers who are interested only in

execution coverage and who do not need the additional information that **lprof** normally provides. For example, if you are developing a test suite and want to find out how much code is actually tested by your test suite, run **lprof** with either the **-x** or **-s** option, depending on the level of detail you want.

Another CSDS profiler you may find useful is **prof**. The **prof** profiler reports the amount of time spent in various parts of a program during execution. The use of **prof** is not required for using **lprof**, but by using these profilers together you can increase the efficiency of **lprof**. The **prof** profiler allows you to identify the most time-consuming parts of a program. By running **lprof** on only those parts of code, you can avoid generating uninformative output while targeting sections of code that need pruning. It is therefore recommended that you use **prof** and **lprof** together.

To use these utilities, you must know how to use CSDS in the UNIX system environment. These utilities do not modify code for you; they enable you to find parts of code that deserve further work on your part. For programmers who have not compiled C code or used CSDS before, the basics are covered in the *AT&T UNIX System V/386 Release 3.2 Programmer's Guide*.

Source Code Control Utilities

A subset of the CSDS utilities, sometimes called the source code control system (SCCS), is specifically designed for source code control. These utilities can be used to record all enhancements and changes to files, along with comments on each version, thus maintaining a history of the changes made. The major SCCS functions include:

- Retrieving any recorded version of a file with comments
- Storing a new version of a file
- Comparing two versions of an SCCS file.

SCCS takes custody of a file, and, when changes are made, identifies and stores them in the file with the original source code and/or documentation. As other changes are made, they too are identified and retained in the file. Each separate set of changes is called a delta. History data can be stored with each version: why the changes were made, who made them, when they were made, etc.

Retrieval of the original or any set of changes is possible. Any version of the file as it develops can be reconstructed for inspection or additional modification.

Here is a list of SCCS commands.

get	Retrieves versions of SCCS files.
unget	Undoes the effect of a **get -e** prior to the file being delta'd.
delta	Applies deltas (changes) to SCCS files and creates new versions.
admin	Initializes SCCS files, manipulates their descriptive text, and controls delta creation rights.
prs	Prints portions of an SCCS file in user-specified format.
sact	Prints information about files that are currently out for edit.
help	Gives explanations of error messages.
rmdel	Removes a delta from an SCCS file. Allows removal of deltas created by mistake.
cdc	Changes the commentary associated with a delta.
what	Searches any UNIX operating system file(s) for all occurrences of a special pattern and prints out what follows that pattern. Useful in finding identifying information inserted by the **get** command.
sccsdiff	Shows differences between any two versions of an SCCS file.
comb	Combines consecutive deltas into one to reduce the size of an SCCS file.
val	Validates an SCCS file.
vc	Is a filter that may be used for version control.

For instructions on how to use SCCS and detailed descriptions of SCCS commands, see the "Source Code Control System" chapter in the *AT&T UNIX System V/386 Release 3.2 Programmer's Guide.*

Extended Terminal Interface

ETI is a set of libraries that promote fast development of screen management applications. The ETI libraries are a software tool that enable you to incorporate screen management and data entry capabilities into your programs. ETI contains the following libraries:

- *Curses/Terminfo Low Level Function Library*: This library consists of routines for writing character-oriented screen management applications independent of the terminal type. Basic routines are provided for writing to a screen, reading from a screen and building windows. Advanced features are used to change screen attributes, draw line graphics and work with more than one terminal. A major new feature is the incorporation of color. You can specify both the background color for each character and the color of the character itself.
- *High-Level Function Libraries*: The high level function libraries are built on top of curses. They consist of functions that create, manipulate, and display panels, forms, and menus.
  - — *Panels*: A panel is a rectangular area containing a curses window that may be displayed in whole or in part on the terminal. Panels provide a depth relationship between curses windows. Panels which are logically below other panels are properly obscured.
  - — *Forms*: A form is a multi-page display that contains a set of fields. These fields may be used for data entry, labels, or messages. You can customize the look and behavior of a form or field. The rich set of form commands includes the following: inter-field and intra-field navigation, field editing, data entry, and validation.
  - — *Menus*: A menu is a display presenting a collection of items. The end-user can select one or more items and this information is available to the application. You can customize the look and behavior of a menu. Menu commands are provided for item navigation, menu scrolling, and item matching.
- *Terminal Access Method (TAM) Transition Library*: The TAM Transition Library enables character mode applications developed for the UNIX PC using TAM to run on other processor/terminal configurations. The library maps TAM calls to curses routines.

Software Features

The CSDS package supports character classification and conversion and international date and time formats. The **ctype**(3C), **ctime**(3C) and **cftime**(4) routines have been modified as described in the following subsections. Also, the dynamic tables of the CSDS components **comp** (compiler) and **as** (assembler) are described. Other CSDS features discussed in the following subsections include referencing a shared library from within a shared library, the #hide and #export directives, checking shared library versions with chkshlib(1), and a proposed C language standard.

ctype(3C)

The classification of characters (what constitutes alphabetic, printable, uppercase or lowercase) varies from language to language. The **ctype**(3C) library routines that are used to classify character-coded integer values have been enhanced to recognize other code sets or classifications. Among these is the routine **setchrclass**(3C), which is a new routine used to initialize the character classification and conversion table. It is invoked at program startup and can be invoked directly from users' programs. This means the character set specific table can change dynamically.

ctime(3C)

The **ctime**(3C) routines allow the user to manipulate date and time formats. Several new library functions (**cftime, ascftime**, and an enhanced **tzset**) have been added to **ctime**(3C). These routines support the following features:

- The ability to specify fractional time zones
- The ability to specify start and end dates and times of alternate time zones
- The ability to specify time and date formats with new format field descriptors
- The ability to specify native language translations of month and weekday names.

cftime(4)

The **cftime**(4) manual page describes how to create language specific files. These files contain detailed information such as full and abbreviated month names, full and abbreviated weekday names, and default local time and date formats.

For more information on how to use these features, see **ctime**(3C), **ctype**(3C), **cftime**(4), and **environ**(5) in the *AT&T UNIX System V/386 Release 3.2 Programmer's Reference Manual.*

Dynamic Tables

Though the C language tends to encourage small functions and source files, some existing applications contained very large source files that failed to compile under previous issues of CSDS because of the fixed size of some tables in the compilation system. In this issue, the tables in the compiler and the assembler are allocated dynamically.

In the compiler, successful compilation is no longer constrained by the number of symbols, the number of cases in a switch, the number of arguments to a function, etc., except as limited by the amount of memory on your machine. Similarly, the assembler's constraint on the number of symbols has been removed.

Referencing a Shared Library from Within a Shared Library

At times you might need to allow one shared library to directly reference routines in another shared library. One way to do this is with imported symbols. Another way is to reference routines in one shared library from another shared library; use the keyword **noload**, with the **#objects** directive in the shared library specification file. When the **#objects noload** directive is used, the **mkshlib** command will search the libraries listed for unresolved references. You will want to use this feature only when you cannot import symbols explicitly.

The #hide and #export Directives

Two directives, **#hide** and **#export**, can be used in the library specification file to control the visibility of external symbols.

Checking Shared Library Versions with chkshlib(1)

The **chkshlib**(1) command allows you to compare versions of shared libraries to see if they are compatible. This command accepts various combinations of executable files, target shared libraries, and host shared libraries as input and tells you if the library versions are compatible, or if the specified executable could have been built by or can run with the specified host or target shared library.

For more information about shared libraries, see the chapter on shared libraries in the *AT&T UNIX System V/386 Release 3.2 Programmer's Guide*. The *AT&T UNIX System V/386 Release 3.2 Programmer's Reference Manual* contains more information about **chkshlib**(1) and **mkshlib**(1).

Proposed Standard for C

As these *Release Notes* were published, no official standard for the C programming language existed. The language accepted by AT&T C compilers follows the definition given in *The C Programming Language* by B. Kernighan and D. Ritchie (Prentice-Hall, 1978). The CSDS package also supports the following extensions.

- Flexnames

  This extension allows variable and function name tokens to be distinct to at least the first 100 characters (rather than the first 8 characters).

- Structure assignments and return values

  This extension allows variables of the same structure type to be assigned to one another. The return value of functions can also be a structure.

- Enumeration types
- Multiple external variable declarations

  This extension makes it possible to have the declaration

  ```
  int i;
  ```

  in multiple source files. All these multiple references resolve to the same address at link edit time.

Currently the X3/J11 task force of the American National Standards Institute (ANSI) is defining a standard for the C language (Draft Proposed American National Standard for Information Systems — Programming Language C, October 1986). The standard proposed by ANSI will allow most current legal C programs to be compiled without any changes. Nevertheless, to ease the possible transition process to the standard, the AT&T C compiler included with CSDS warns about the use of some constructs that may not be legal in the future or may cause portability problems. The following are examples of such constructs.

- Declarations, such as,

  ```
  int  i;
   static int i;
  ```

  produce the warning message

  ```
  warning: i previously declared extern, becomes static.
  ```

- Structure definitions missing semicolons, such as

  ```
  struct x {
    int i
  }
  ```

  produce the warning message

  ```
  warning: syntax requires ; at end of struct/union decl
  ```

Installation Notes

The following text describes the space dependencies and version control as it relates to the installation of the SDS package. For complete installation procedures, see the *Operations/System Administration Guide*.

Space Dependencies

The SDS package is installed using the *installpkg*(1) command. The *installpkg*(1) command checks to determine that sufficient free space is available in the **root** and **/usr** file systems. You need approximately 7,900 blocks (512-byte blocks) of memory to install the SDS package.

Version Control

The C software development set portion of the SDS package uses a per file method of version control. If the file being installed already exists on the system and has a release number greater than the file belonging to the package being installed, the existing file will not be overwritten. Files without valid release information are assumed to be older than those belonging to the package being installed.

Verification

After installing SDS, verify the correct SDS version (4.1.5) by using cc -V.

Software Notes

This section offers some tips on using the SDS package and some software tips that enhance the usability of the package.

1. Functions of type *float* or *double* need to be declared in scope whether or not their return values are being used.
2. Elements of type *char* will be sign extended. For zero extension, *unsigned char* must be used.
3. If you are compiling your programs with the -g option enabled so that you can do debugging, it is advisable **NOT** to use the -O option as well. In some cases, the two options invoked jointly will produce multiply defined labels. In addition, you should not use -O when compiling -ql because this in turn turns on the -g flag.
4. The default response to the invalid operation, divide by zero, and overflow exceptions is to take a trap. This behavior may be altered by using the **fpsetmask(3)** function.
5. When an *Intel* 80287 co-processor is installed, use of denormalized floating point numbers results in a core dump. The problem is that the 80287 chip does not normalize a denormal number when it is loaded and produces an invalid operation exception when a denormal number is stored to memory. If such problems are encountered, one work-around is to enable the denormalized operand exception and provide a signal handler which normalizes a denormal number. This signal handler must also recognize any other enabled traps (signals).
6. Without an *Intel* 80287 or 80387 coprocessor installed, the floating point emulator incorrectly returns 0 rather than NaN for any operation on NaN.
7. The IEEE 754 standard for floating point (IEEE Standard for binary Floating-Point Arithmetic, ANSI/IEEE Std. 754-1985) allows several different methods for detecting overflow. As a result, you should not rely on a particular implementation to signal overflow for a particular operation.
8. Floating point comparisons where one operand is an NaN always result in an invalid operation exception. This is because the *Intel* 80287 lacks an instruction to make this comparison without getting the exception.

9. **dis(1)** and **sdb(1)** do not recognize the *Intel* 80387 specific instructions.

10. **pipe(2)** — The documentation states that the maximum number of bytes in a pipe (PIPE_MAX) is defined to be 5120. The system sets PIPE_MAX to 10240.

11. **ioctl(2)** — The **V_ADDBAD** command (notifies the device drivers of bad sectors) in **ioctl(2)** updates only the table currently in memory and does not update the table on the hard disk. Therefore, all the changes made using **ioctl(2)** with **V_ADDBAD** will be lost when the system is rebooted. Also, if an assigned alternate sector goes bad, there is no way to recover.

12. **ioctl(2)** — The **V_GETPARMS** command in **ioctl(2)** returns the incorrect number of sectors for a 360KB device. The number of sectors reported is 1440; however, the correct value is 720.

13. The Graphics Programming Utilities (GPU) has been renamed extended terminal interface (ETI).

14. A new function, **gethz()**, is added to **libc** that gets the HZ value from the environment.

15. The **-s** option is added to the symbolic debugger (**sdb**). The system will not catch the trap specified by the **-s** option. For example, if you specified **sdb -s2**, it will not trap on error number 2.

16. The **-Zp[1|2|4]** option is added to **cc**. This option packs structure members in memory. Normally, structure members are aligned as follows: items of type **char** are byte-aligned, items of type **short** are aligned on 2-byte boundaries, and all other types of structure members are word-aligned. Specifying an option to **-Zp** will force alignment on the given byte boundary. If no option is used with **-Zp**, structure members will be packed on 1-byte boundaries. The alignment may be altered with the **#pragma pack** preprocessor directive.

17. The **-x** option is added to **convert**. This is required to convert a *Xenix* archive. Using this option will convert the general archive structure but leave archive members unmodified.

Compatibility

This section describes the changes made in this issue of the SDS package that may have an effect on the compatibility of your programs.

The Compiler and cc

The following compatibility notes concern changes made to the CSDS **cc** command or the compiler, **comp**, in this issue of the SDS. These notes apply only if you are porting C programs compiled on an AT&T compilation system (release number less than 4.1) for a different machine.

- The **-B** and **-t** options have been removed from the **cc** command. Previous releases printed a warning message that these options would disappear.
- The handling of aggregate initialization has been changed to conform to the definition given by Kernighan and Ritchie. Initialization where all braces are specified or where only the outermost braces are specified continues to work as before.
- **cc** and **comp** can no longer take the address of a label.

  The following illegal C code will no longer compile:

```
f(){
    int i;
lab:
    i = (int) &lab;
}
```

- Bad structure code, such as the following, is disallowed:

  taking the address of the return value of a function which returns a structure:

```
pst = &(stcall());
```

  using a function return value as an L-value:

```
stcall() = *pst;
```

  taking the address of a structure assignment:

```
pst = &(st1=st2);
```

cpp

The following change was made to **cpp** in this issue of the SDS.

- A missing or invalid macro name in `ifdef`, `ifndef`, `undef`, or `define` is now a fatal error.

  For example:

```
#ifdef 202
#undef
#undef 1abc
```

Changes in C Library Functions

The following list describes changes made to functions in the C library in this issue of the SDS.

ctime(3C) An **a.out** compiled with previous versions of the **ctime** functions when used with some new legal TZ values will give unexpected results.

ctime(3C) **ctime** now defaults to GMT if TZ is not set.

In previous releases it defaulted to EST.

fgets(3S) A call to **fgets** on a write-only file returns NULL. In earlier releases, **fgets** always returned the address of the buffer passed to it.

fread(3S), **fwrite**(3S)

The **fread** and **fwrite** functions return zero when **size** is zero or huge.

In an earlier release, these two functions always returned **nitems**. **size** and **count** are multiplied to give the number of bytes to be transferred. If the result is larger than the remaining bytes of the file or is not representable within the precision of an integer, fewer items will be read than requested and the number of items actually read will be returned.

scanf(3S) Calls to **scanf** now return EOF on end-of-file. In an earlier release, **scanf** erroneously returned zero.

Environment Variables

The variables CFTIME, CHRCLASS, and LANGUAGE are environment variables in CSDS. Setting them may cause C library functions to change their behavior. Also, the TZ environment variable may be interpreted differently. The following table lists the library functions affected by these variables.

Function	Environment Variables
ctime	TZ
isalnum	CHRCLASS
isalpha	CHRCLASS
iscntrl	CHRCLASS
isdigit	CHRCLASS
isgraph	CHRCLASS
islower	CHRCLASS
isprint	CHRCLASS
ispunct	CHRCLASS
isupper	CHRCLASS
localtime	TZ
tolower	CHRCLASS
toupper	CHRCLASS

The mkshlib Command

Uninitialized external variables (common symbols) are illegal in a shared library. Previously, the use of common symbols was discouraged by both the documentation and a **mkshlib** warning message. This warning message is now a fatal error.

Future Directions

This section describes areas of the SDS product that are likely to change in future releases.

1. It is likely that some of the functions defined in **libPW** will be removed in a future release of this product. If you have any code that relies on **libPW**, AT&T recommends that you reimplement it using existing functions in the standard C library or that you retain copies of the **libPW** functions that you need.
2. The **list**(1) command will be removed in a future release of this product.
3. AT&T expects, in a future release of the SDS, to support the ANSI Standard for the C language once the standard is accepted. That standard introduces the keywords **const**, **signed**, and **volatile**. Programmers should therefore avoid using these words as identifiers in programs.
4. A major feature of the extended terminal interface (ETI) is the ability to turn on and off any of several video attributes, such as bold, dim, blinking, underlining, reverse video, and others. Future enhancements of ETI will include additional video attributes that enable your programs to use the color capabilities of a wide range of terminals.
5. In keeping with AT&T's ongoing internationalization of the UNIX system, future users will be able to use ETI with keyboards using foreign language character sets, such as Kanga.

Documentation

Essential documentation is provided with the SDS software package when purchased. Additional sets of the Software Development documentation (of which these *Release Notes* are a part) are available and can be ordered. See the Product Overview/Documentation Roadmap for more details. The Product Overview/Documentation Roadmap can be ordered separately by using the 9-digit number 999-300-527.

Documentation Updates

The following change pages reflect last minute changes to the UNIX System V/386 Release 3.2 documentation. These change pages should be inserted into the *Programmer's Reference Manual*.

AT&T UNIX SYSTEM V/386
RELEASE 3.2
PROGRAMMER'S REFERENCE MANUAL
UPDATES

This update involves the following actions:

1. **ACTION:** Replace RMDIR(2) pages 1 and 2 with the new pages.
2. **ACTION:** Replace SEMGET(2) pages 1 and 2 with the new pages.
3. **ACTION:** Replace UNLINK(2) pages 1 and 2 with the new pages.

NAME

rmdir – remove a directory

SYNOPSIS

int rmdir (path)
char *path;

DESCRIPTION

rmdir removes the directory named by the path name pointed to by *path*. The directory must not have any entries other than "." and "..".

The named directory is removed unless one or more of the following is true:

[EINVAL]	The current directory may not be removed.
[EINVAL]	The "." entry of a directory may not be removed.
[EEXIST]	The directory contains entries other than those for "." and "..".
[ENOTDIR]	A component of the path prefix is not a directory.
[ENOENT]	The named directory does not exist.
[EACCES]	Search permission is denied for a component of the path prefix.
[EACCES]	Write permission is denied on the directory containing the directory to be removed.
[EBUSY]	The directory to be removed is the mount point for a mounted file system.
[EROFS]	The directory entry to be removed is part of a read-only file system.
[EFAULT]	*Path* points outside the process's allocated address space.
[EIO]	An I/O error occurred while accessing the file system.
[ENOLINK]	*Path* points to a remote machine, and the link to that machine is no longer active.
[EMULTIHOP]	Components of *path* require hopping to multiple remote machines.

In addition, a directory will not be removed when all of the following is true:

the parent directory has the sticky bit set
the parent directory is not owned by the user
the target directory is not owned by the user
the target directory is not writable by the user
the user is not super-user

SEE ALSO

mkdir(2).

rmdir(1), rm(1), and mkdir(1) in the *User's/System Administrator's Reference Manual*.

DIAGNOSTICS

Upon successful completion, a value of 0 is returned. Otherwise, a value of −1 is returned and *errno* is set to indicate the error.

NAME

semget – get set of semaphores

SYNOPSIS

#include <sys/types.h>
#include <sys/ipc.h>
#include <sys/sem.h>

int semget (key, nsems, semflg)
key_t key;
int nsems, semflg;

DESCRIPTION

The *semget* system call returns the semaphore identifier associated with *key*.

A semaphore identifier and associated data structure and set containing *nsems* semaphores [see *intro*(2)] are created for *key* if one of the following is true:

Key is equal to **IPC_PRIVATE**.

Key does not already have a semaphore identifier associated with it, and (*semflg* & **IPC_CREAT**) is "true".

Upon creation, the data structure associated with the new semaphore identifier is initialized as follows:

Sem_perm.cuid, **sem_perm.uid**, **sem_perm.cgid**, and **sem_perm.gid** are set equal to the effective user ID and effective group ID, respectively, of the calling process.

The low-order 9 bits of **sem_perm.mode** are set equal to the low-order 9 bits of *semflg*.

Sem_nsems is set equal to the value of *nsems*.

Sem_otime is set equal to 0 and **sem_ctime** is set equal to the current time.

The data structure associated with each semaphore in the set is not initialized. The function *semctl* with the command *setval* or *setall* can be used to initialize each semaphore.

The *semget* system call fails if one or more of the following is true:

[EINVAL] *Nsems* is either less than or equal to zero or greater than the system-imposed limit.

[EACCES] A semaphore identifier exists for *key*, but operation permission [see *intro*(2)] as specified by the low-order 9 bits of *semflg* would not be granted.

[EINVAL] A semaphore identifier exists for *key*, but the number of semaphores in the set associated with it is less than *nsems*, and *nsems* is not equal to zero.

[ENOENT] A semaphore identifier does not exist for *key*, and (*semflg* & **IPC_CREAT**) is "false".

[ENOSPC] A semaphore identifier is to be created, but the system-imposed limit on the maximum number of allowed semaphore identifiers system wide would be exceeded.

[EEXIST] A semaphore identifier exists for *key*, but [(*semflg* & **IPC_CREAT**) and (*semflg* & **IPC_EXCL**)] are "true".

SEE ALSO

intro(2), semctl(2), semop(2).

DIAGNOSTICS

Upon successful completion, a non-negative integer, namely a semaphore identifier, is returned. Otherwise, a value of –1 is returned, and *errno* is set to indicate the error.

NAME

unlink – remove directory entry

SYNOPSIS

int unlink (path)
char *path;

DESCRIPTION

unlink removes the directory entry named by the path name pointed to by *path*.

The named file is unlinked unless one or more of the following is true:

[ENOTDIR]	A component of the path prefix is not a directory.
[ENOENT]	The named file does not exist.
[EACCES]	Search permission is denied for a component of the path prefix.
[EACCES]	Write permission is denied on the directory containing the link to be removed.
[EPERM]	The named file is a directory and the effective user ID of the process is not super-user.
[EBUSY]	The entry to be unlinked is the mount point for a mounted file system.
[ETXTBSY]	The entry to be unlinked is the last link to a pure procedure (shared text) file that is being executed.
[EROFS]	The directory entry to be unlinked is part of a read-only file system.
[EFAULT]	*Path* points outside the process's allocated address space.
[EINTR]	A signal was caught during the *unlink* system call.
[ENOLINK]	*Path* points to a remote machine and the link to that machine is no longer active.
[EMULTIHOP]	Components of *path* require hopping to multiple remote machines.

A file will not be unlinked when all of the following is true:

the parent directory has the sticky bit set
the file is not writable by the user
the user does not own the parent directory
the user does not own the file
the user is not super-user

When all links to a file have been removed and no process has the file open, the space occupied by the file is freed and the file ceases to exist. If one or more processes have the file open when the last link is removed, the removal is postponed until all references to the file have been closed.

SEE ALSO

close(2), link(2), open(2).

rm(1) in the *User's/System Administrator's Reference Manual*.

DIAGNOSTICS

Upon successful completion, a value of 0 is returned. Otherwise, a value of −1 is returned and *errno* is set to indicate the error.

Appendix B: Installation Diskette Files

Appendix B: Installation Diskette Files

C Software Development Set Utility Package, Contents of 4 Diskettes

Size
Files
Name
Install
Remove
bin
bin/ar
bin/cc
bin/gencc
bin/as
bin/ld
bin/cprs
bin/conv
bin/dis
bin/dump
bin/list
bin/lorder
bin/mkshlib
bin/chkshlib
bin/nm
bin/size
bin/strip
bin/convert
bin/make
lib
lib/libld.a
lib/comp
lib/cpp
lib/optim
lib/cm4defs
lib/libPW.a
lib/crt0.o
lib/crt1.o
lib/crtn.o
lib/mcrt0.o
lib/mcrt1.o
lib/pcrt1.o
lib/libc.a
lib/libm.a
lib/libx.a
lib/libc_s.a
lib/basicblk
lib/pcrt0.o
etc
etc/install
usr
usr/bin
usr/bin/cflow
usr/bin/ctrace
usr/bin/ctcr
usr/bin/ctc
usr/bin/cxref
usr/bin/lex
usr/bin/lint
usr/bin/lprof
usr/bin/m4
usr/bin/prof
usr/bin/regcmp
usr/bin/admin
usr/bin/cdc
usr/bin/comb
usr/bin/delta
usr/bin/get
usr/bin/prs
usr/bin/rmdel
usr/bin/sact

usr/bin/unget
usr/bin/val
usr/bin/vc
usr/bin/what
usr/bin/sccsdiff
usr/bin/sdb
usr/bin/yacc
usr/bin/tsort
usr/bin/cb
usr/bin/cscope
usr/lib
usr/lib/libp
usr/lib/libp/libc.a
usr/lib/libp/libm.a
usr/lib/libp/libmalloc.a
usr/lib/libp/libx.a
usr/lib/ctrace
usr/lib/ctrace/runtime.c
usr/lib/libcrypt_i.a
usr/lib/llib-lc
usr/lib/libg.a
usr/lib/libl.a
usr/lib/libmalloc.a
usr/lib/llib-lmalloc.l
usr/lib/liby.a
usr/lib/dag
usr/lib/lpfx
usr/lib/help
usr/lib/help/lib
usr/lib/help/lib/help2
usr/lib/help/lib/help
usr/lib/help/ad
usr/lib/help/bd
usr/lib/help/cb
usr/lib/help/cm
usr/lib/help/cmds
usr/lib/help/co
usr/lib/help/de
usr/lib/help/default
usr/lib/help/ge
usr/lib/help/he
usr/lib/help/prs
usr/lib/help/rc
usr/lib/help/un
usr/lib/help/ut
usr/lib/help/vc
usr/lib/nmf
usr/lib/flip
usr/lib/xpass
usr/lib/xcpp
usr/lib/llib-port
usr/lib/llib-lc.ln
usr/lib/llib-lm
usr/lib/llib-port.ln
usr/lib/yaccpar
usr/lib/llib-lm.ln
usr/lib/lex
usr/lib/lex/ncform
usr/lib/lex/nrform
usr/lib/lint1
usr/lib/lint2
usr/lib/basicblk
usr/lib/libprof.a
usr/include
usr/include/a.out.h
usr/include/aouthdr.h
usr/include/ar.h
usr/include/assert.h
usr/include/core.h
usr/include/ctype.h
usr/include/dial.h
usr/include/dirent.h
usr/include/errno.h
usr/include/fatal.h
usr/include/fcntl.h
usr/include/filehdr.h
usr/include/ftw.h
usr/include/grp.h
usr/include/ieeefp.h
usr/include/ldfcn.h

usr/include/limits.h
usr/include/linenum.h
usr/include/macros.h
usr/include/malloc.h
usr/include/math.h
usr/include/memory.h
usr/include/mnttab.h
usr/include/mon.h
usr/include/nan.h
usr/include/nlist.h
usr/include/nsaddr.h
usr/include/nserve.h
usr/include/poll.h
usr/include/prof.h
usr/include/pwd.h
usr/include/regexp.h
usr/include/reloc.h
usr/include/rje.h
usr/include/scnhdr.h
usr/include/sd.h
usr/include/search.h
usr/include/setjmp.h
usr/include/sgtty.h
usr/include/signal.h
usr/include/stand.h
usr/include/stdio.h
usr/include/storclass.h
usr/include/string.h
usr/include/stropts.h
usr/include/strselect.h
usr/include/syms.h
usr/include/sys.s
usr/include/termio.h
usr/include/time.h
usr/include/tp_defs.h
usr/include/unistd.h
usr/include/ustat.h
usr/include/utmp.h
usr/include/values.h
usr/include/varargs.h

DOC0043-2Y