Lucent Technologies
Bell Labs Innovations

CommKit Host Interface Release 4.4 AT&T 3B2 RISC Computer Systems Installation and Administration Guide

255-110-127
Issue 3
September 1995

Additional Information

Federal Communications Commission Statement

This equipment generates, uses, and can radiate radio frequency energy and if not installed and used in accordance with the instruction manual, may cause interference to radio communications. It has been tested and found to comply with the limits for a Class A computing device pursuant to Subject J or Part 15 of FCC rules.

Trademarks

CommKit® is a registered trademark of AT&T.
Datakit® is a registered trademark of AT&T.
UNIX® is a registered trademark of Novell, Incorporated in the United States and other countries, licensed exclusively through X/Open Company, Ltd.
Polywater® is a registered trademark of American Polywater Company.
Hydralube Blue® is a registered trademark of Arnco Equipment Company.
Imagen is a registered trademark of Imagen Corp.
PostScript is a registered trademark of Adobe Systems Inc.

Ordering Information

Additional copies of this document can be ordered by calling:

U.S.A.: 1-800-432-6600 Canada: 1-800-255-1242 Other Areas: 1-317-352-8557

or, by writing to: AT&T Customer Information Center
Attn: Customer Service Representative
P.O. Box 19901
Indianapolis, IN 46219

Table of Contents

1 Introduction

5 Compatibility

6 Manual Pages

Figures

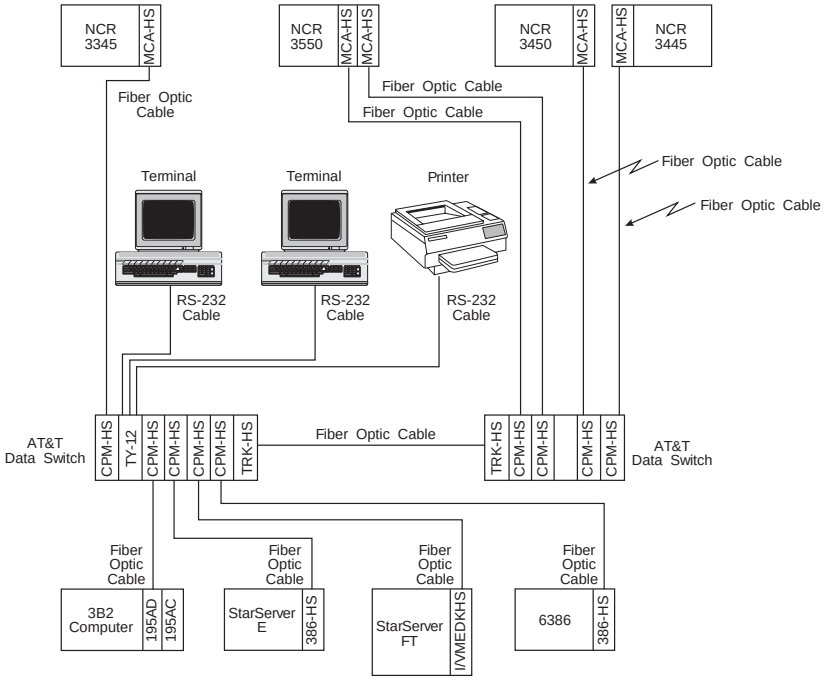
Tables

1 Introduction

General

This Installation and Administration Guide describes how to install and administer the AT&T CommKit Host Interface on the AT&T 3B2 RISC Computer System. The *host interface* consists of hardware and CommKit Host Interface software on your computer system that provides the high-speed data link to an AT&T data switch node. Figure 1-1 shows a typical data switch network.

Figure 1-1: Data Network



Note: Your network may include BNS-1000, BNS-2000, Datakit VCS, and Datakit II VCS data switches. However, the host interface must be connected to a BNS or Datakit II VCS. Throughout this document, *data switch* refers to the specific data switch to which you are connecting.

Description

The host interface software and hardware allow you to connect your computer to an AT&T data switch. The interface consists of:

1. Software installed on the computer
2. A host interface module installed in the computer
3. A CPM-HS module installed in the data switch.

The host interface module and CPM-HS are connected with an optical fiber link.

Host Access

Most host computers have a limited number of RS-232 connections available; extra ports may become available in groups of 8 or 16 with the addition of port modules. This can tie up multiple I/O module slots in the host. Using CommKit Host Interface software, you can configure up to 512 channels per fiber interface module. Each channel acts as an RS-232 connection; a user can log into these channels from a terminal or other host, or data can be sent to an output device such as a printer.

As needs change, the number of channels the CommKit Host Interface software supports can be changed with software commands.

User Features

The host interface offers features valuable to most users:

Local/Remote Login

Users can log on to their local systems or remote systems within a wide area network (WAN). Users are no longer locked into a single system or local area network (LAN).

File/Directory Transfer

Users can transfer files and directories with a single command. Unlike ***uuto*** which only sends one file at a time, users can easily transfer an individual file or a complete directory structure (with all sub-directories and files) using a single CommKit Host Interface software command. CommKit Host Interface software allows the sending and retrieving of files and directories as a single-step process. And, CommKit Host Interface software allows users to transfer directories across the network with the same command.

Remote Execution

Users can execute commands on local or remote systems with a single command. The commands can be standard UNIX system applications, customized user programs, or shell scripts.

Administrative Features

The host interface offers several features to help system administrators.

Security

CommKit Host Interface software has been designed with security in mind. It provides user ID security or transparent user ID mapping security, which prevents the same user ID on one system from masquerading as another user on a remote system. User commands for file transfer honor and preserve directory and file permissions. Additionally, administrators can include/exclude specific users and groups from access to a host computer over the host interface.

Services

The CommKit Host Interface software allows the user access to ***listener*** and **dkserver** services. Access to the **dkserver** service depends on permissions (originating group security, refer to Chapter 3) and endpoint (service) addresses.

Figure 1-2: Logical Connectivity

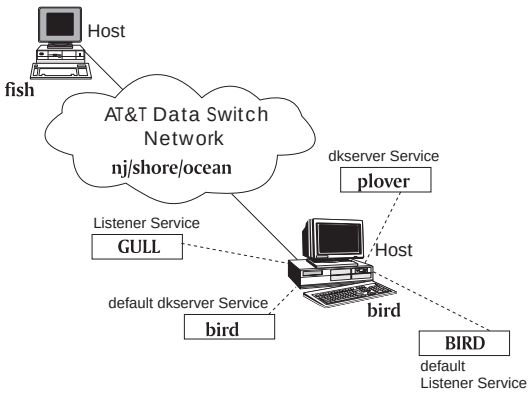


Figure 1-1 illustrates the physical connectivity of the CommKit Host Interface within a data switch network. A logical connectivity schematic is shown in Figure 1-2. To use the CommKit Host Interface services, the administrator must understand the logical connections between host names, server names, and originating group security.

The host name (***uname***) can connect to listener or other services by defining a different server name for each service. Consider the table below:

Orig Host	Orig Group	Dest Host	Dest Service	Address	Comment
fish	nj/shore/fish	bird	dkserver	nj/shore/bird	default dkserver
fish	nj/shore/fish	bird	listener	nj/shore/BIRD	default listener
fish	nj/shore/fish	bird	dkserver	nj/shore/plover	plover dkserver
fish	nj/shore/fish	bird	listener	nj/shore/GULL	GULL listener
bird	nj/shore/bird	fish	dkserver	nj/shore/fish	default dkserver

The originating group (of the calling endpoint) is used for security by the **dkserver** receiving the call. The name used when starting a **dkserver** or ***listener*** on a receiving host must match an address configured for the CPM-HS or the host.

The CommKit Host Interface services are defined in detail in Chapter 4.

Installation and Removal

Administrators will find it easy to install and remove CommKit Host Interface software because it uses standard facilities and follows conventions established by the UNIX system.

Configuration and Maintenance

Administrators can reconfigure the number of virtual circuits into a host computer with a special **dkdaemon** application. New services and servers can be added or removed quickly and easily. The administrator can also monitor the status of the interface and perform diagnostics on the hardware.

Print Spoolers

Printers can be connected to the network in a variety of ways. In some networks, administrators connect a serial printer to a data switch TY port and access the printer from a number of hosts.

uucp Services

The UNIX Basic Networking Utilities (BNU) can be configured to use CommKit Host Interface software. This allows BNU commands (e.g., ***cu*** and ***uucp***) to use the network. BNU can be configured with one or more error detecting protocols depending on the network topography.

File Sharing

The remote file sharing (RFS) feature can be configured to use the network. This allows administrators to ***mount*** UNIX system files from other systems and use these files on their own system.

Note: RFS is not supported by the NCR UNIX System.

Development Features

The host interface provides the following features to software developers who want to make their applications available over networks.

Fiber Optic Link

The customer has access to a reliable fiber optic link that offers high throughput and the necessary number of virtual circuits for most applications.

General Server

Developers can administer applications with a general purpose server, **dkserver**. Applications can be written with minimal network considerations. The server manages network protocol and network security issues.

High Performance Application Library

The CommKit Host Interface software library, **libdk.so**, is delivered as a dynamic shared library object module. By writing applications that use the services of **libdk.so**, most of the details of the interface can be hidden. These library functions are described in Chapter 5.

Network Independent Application Library

CommKit Host Interface software also supports Transport Layer Interface (TLI). Using this library allows development of applications that can run over other networks without modification.

Document Overview

This guide is designed for the system administrator and is organized as follows:

- *Introduction*.
- *Installation and Removal* – step-by-step hardware and software procedures and procedures for configuring the CPM-HS on the data switch node.

- *Control Tables* – detailed description of building and editing the **dkhosts, dkdotab, srvtab**, and **dkuidtab** control tables.
- *Administration* – instructions for changing the hardware configurations, administering ***uucp*** and remote file sharing (RFS), a summary of network error messages, and printer sharing procedures.
- *Compatibility* – information for the current CommKit Host Interface software release operating with UNIX System V Release 4.0.
- *Manual Pages.*

Reference Documentation

Tables 1-1 through 1-3 list the UNIX system, data switch, and the host interface documentation that are referenced throughout this document.

Table 1-1: UNIX System Documentation

Title	ISBN
UNIX System V Programmer's Guide: STREAMS	013947003-4
UNIX System V Programmer's Reference Manual	013947029-8
UNIX System V System Administrator's Reference Manual	013947011-5
UNIX System V System Administrator's Guide	013947086-7
UNIX System V Network User's and Administrator's Guide	013933813-6
UNIX System V User's Reference Manual	013947037-9

Table 1-2: AT&T Host Interface Documentation

Document	Select Code
Datakit II VCS Host Interface for AT&T 6386 WGS Installation and Administration Guide	255-110-105
CommKit Host Interface Release 4.0/4.1 AT&T 386/486 Computers Installation and Administration Guide	255-110-115

Table 1-2: continued on next page

Table 1-2: *Continued*

CommKit Host Interface Release 3.3 StarServer FT Installation and Administration Guide	255-110-123
CommKit Host Interface Release 4.2 NCR Series 3000 Computers Installation and Administration Guide	255-110-124
CommKit Host Interface Release 4.3 StarServer FT Installation and Administration Guide	255-110-125
CommKit Host Interface Release 4.4 AT&T 3B2 RISC Computer Systems Installation and Administration Guide	255-110-127
Datakit II VCS Internal Interface Specification	700-283

Table 1-3: AT&T Data Switch Documentation

Title	Select Code
BNS-1000 Administrator's Guide	255-180-500
BNS-1000 Cabling Guide	255-180-606
BNS-1000 Commands Reference	255-180-200
BNS-1000 Installation Guide	255-180-100
BNS-1000 Messages Reference	255-180-201
BNS-2000 Node Reference	255-181-220
BNS-2000 System Description	255-181-110
Datakit II VCS Administrator's Guide	255-113-500
Datakit II VCS Commands Reference	255-113-200
Datakit II VCS Installation Guide	255-113-100
Datakit II VCS Planning Guide	255-183-120
Datakit II VCS Messages Reference	255-113-201

Glossary of Terms

The following list of acronyms and abbreviations are used in this document:

General Terms

BNU	Basic networking utilities	**SAF**	Service access facility
ESD	Electrostatic discharge	**ST**	Standard tip
LAN	Local area network	**TLI**	Transport layer interface
NLPS	Network layer provider service	**TPI**	Transport provider interface
NSU	Networking support utilities	**Tx**	Transmitter
RFS	Remote file sharing	**URP**	Universal receiver protocol
Rx	Receiver	**WAN**	Wide area network

Display Conventions

This document contains sample displays that will help you understand the described procedures.

Note: The displays in this document may differ from those on your terminal screen because of product improvements made after this document was completed. Your terminal screen accurately reflects the software on your computer.

In command discussions, user input and computer response examples are shown as follows:

```
System generated responses and messages are displayed in constant width
typeface with variable messages in italic typeface as shown here.

User input displayed on screen is displayed in constant width bold
typeface as shown here.

User input of the Delete and Enter keys are shown as follow: [Delete] [Enter]

Simultaneous entry of the control key with another key is shown as follows: [Ctrl-d]
```

The dollar sign is the default UNIX system prompt. The symbol # is the system prompt for **root**. In some cases this symbol can indicate a comment line that you may see when viewing files.

The data switch control computer system prompt may be either `CC>` or `CC0>` depending on the release.

Format Conventions

Throughout this document UNIX and Datakit VCS commands are shown in ***bold italic*** font; CommKit Host Interface commands, process names, program names, and control tables are shown in **bold** type; file and directory names are shown in *italic* type. References to man pages are shown in *italic* typeface [e.g., *dkserver*(1M)].

CommKit Host Interface Software Release/UNIX System Version Information

Table 1-4 lists the current CommKit Host Interface software releases and the supported UNIX system version supported for each release.

Table 1-4: CommKit Host Interface Software Release/UNIX System Version

CommKit Host Interface Software Release	Supported UNIX System Version
4.0v1, 4.0v2	SVR4.0 versions 1 and 2
4.0v3, 4.0v4	SVR4.0 versions 1 and 2.1; SVR4.2 version 1
4.1v1, 4.1v2, 4.1v3	SVR4.0 versions 3.0 and 3.1
4.2v1, 4.2v2, 4.2v3, 4.2v4	SVR4.0 version 3.0, MP-RAS 2.0, 2.01, 2.02, 2.03, 3.00
4.3v1, 4.3v2	SVR4.0 B22ipm10, B22ipm31, B32
4.4v1, 4.4v2, 4.4v3	SVR4.0.3, 4.0.4

Equipment Description

To add your computer to the data switch network, the following are required:

Computer System:

- Host interface module
- Cartridge Tapes
- Ground Clips
- Cable tie

Data switch node:

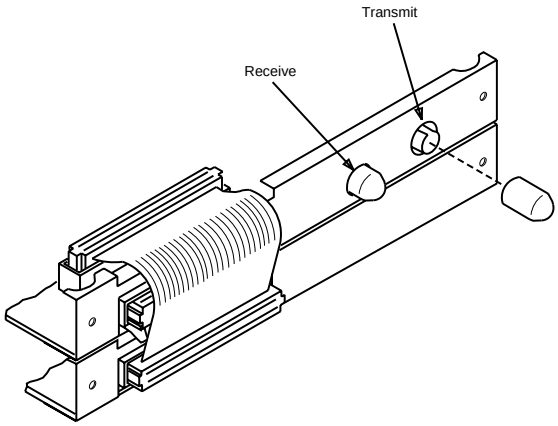
- CPM-HS module
- Paddle board
- Fiber optic cable

Host Interface Module

The host interface module – which is to be installed in the 3B2 computer – contains the firmware required for the 3B2 computer to interface with the data switch node. Refer to Figure 1-3.

A cable tie is used to hold the cables securely to the 3B2 computer cabinet to prevent damage to the hardware and fiber optic cable. Ground clips are used to ensure the host interface module is properly grounded.

Figure 1-3: Host Interface Module



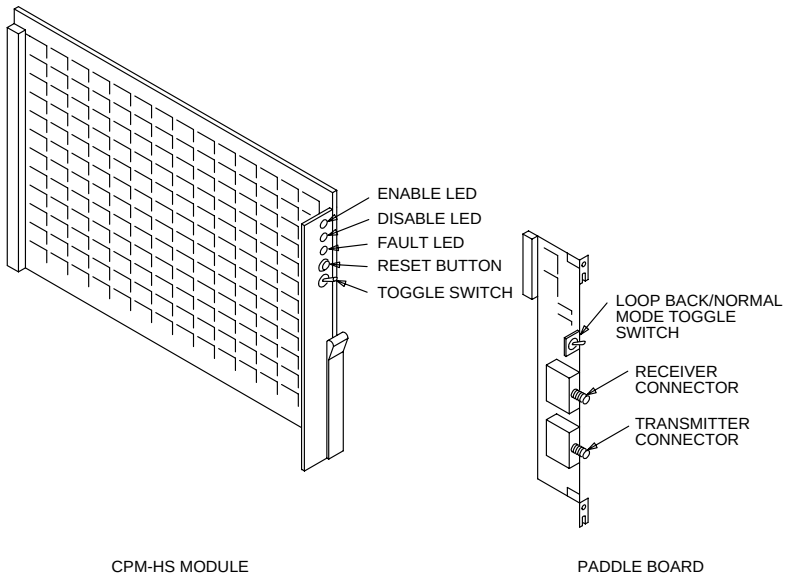
CPM-HS Module and Paddle Board

The CPM-HS module (Figure 1-4) is installed in one of the slots at the front of the data switch node. The CPM-HS module faceplate has a toggle switch, a reset button, and three LEDs.

- The toggle switch enables / disables the computer from the network
- The reset button resets the CPM-HS module after a fault condition occurs
- The three LEDs give the state of the CPM-HS module.

The paddle board is installed in the same numbered slot as the CPM-HS module at the rear of the data switch. The paddle board has transmit and receive connectors for establishing a fiber optic link between the computer system and the data switch node. The paddle board contains a loopback / normal mode toggle switch. This switch should be left in the normal position; the loopback position is used for local loopback testing.

Figure 1-4: CPM-HS Module and Paddle Board



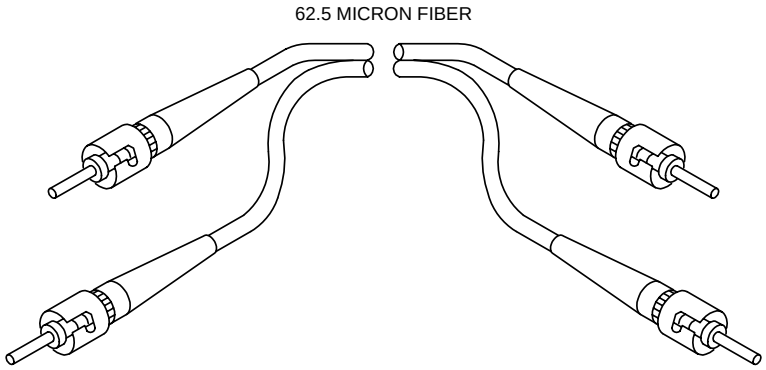
Fiber Optic Cable

Fiber optic cable provides the connection between the computer system and the data switch node. This thin, lightweight, flexible cable has many advantages over an RS-232 cable, such as:

- Greater bandwidth
- Less noise or crosstalk
- Can be used for longer distances
- More security.

Figure 1-5 shows the fiber optic cable required to connect the computer to the data switch node. This fiber optic cable requires standard tip (ST) connectors on both ends.

Figure 1-5: Fiber Optic Cable (Dual Cable Shown)



Cartridge Tapes

The host interface package cartridge tapes hold the software that contains the network interface commands, programs, diagnostics, and utilities that are necessary for the host interface to work on your computer system.

The TLI Support software provides the UNIX system kernel support which allows you to develop or utilize applications that rely on UNIX system TLI. Examples of applications that rely on TLI are **RFS** and **uucp** (specifically the **e** protocol).

Software Certificate

This release of the CommKit Host Interface software is copy-protected and requires a Software Certificate Number and Software Key before it can be used. The certificate included with your software contains your certificate number and instructions for obtaining your personalized software key.

Customer Assistance

AT&T offers a complete package of support services to customers needing assistance with installation, administration, operation, and maintenance of AT&T Data Networking products. These support services are available on either a *Contract* or a *Time-and-materials* basis.

For more information on these service offerings, or to establish a service account, contact your AT&T account representative, or call 1-800-WE2-CARE during our standard business hours (8AM – 5PM Eastern Time, Monday through Friday, excluding holidays) and ask for the Data Networking Services Operations department.

When you need assistance, call 1-800-WE2-CARE during the coverage period selected when you established your service account. You will need to provide your Service Account Number and identify the specific AT&T Data Networking product for which you need assistance.

Outside the United States, AT&T Data Networking products and support services are provided by AT&T subsidiaries and authorized value-added resellers. For more information, contact your sales representative.

Manual Pages

The manual pages are provided at the end of this document as a quick reference to the CommKit Host Interface software commands.

The manual pages are divided into four sections each consisting of entries arranged in alphabetical order.

Section	Classification	Description
1	1C 1M	Communications Maintenance
3	3X	Miscellaneous Routines
4	4	File Formats
7	7	Special Files

- **Section 1C Application Programs** – contains communication programs which reside in the directory */opt/dk/bin*.
- **Section 1M System Maintenance Commands** – contains system maintenance programs which reside in */opt/dk/sbin*.
- **Section 3X Subroutines** – describes the binary versions residing in system libraries in the directory */usr/lib*.
- **Section 4 File Formats** – describes the structure of kinds of various files: the possible contents of fields; the possible fields in each line.
- **Section 7 Special Files** – describes the characteristics of each system file that refers to an input/output device. The names in this section generally refer to device names for the hardware, rather than to the names of the special files.

Format

All entries are based on a common format; some entries may not contain all parts. The parts are listed below.

NAME — gives the name of the entry and briefly states its purpose.

SYNOPSIS — summarizes the use of the command being described. The conventions used are:

Bold typeface	Enter these strings literally as they appear.
Italic	Substitutable argument prototypes, program names, files, and directories.
[ ]	These brackets indicate an optional argument prototype.
{ }	These brackets around a series of argument prototypes indicate **one** of the arguments is mandatory. When an argument prototype is given as "name" or "file," it always refers to a *file* name.
–, +, or =	These symbols indicate some sort of flag argument even if they appear in a position where a filename could appear.

DESCRIPTION — describes the command.

EXAMPLE(S) — gives examples of use.

FILES — gives the file names that are built into the program.

SEE ALSO — gives additional references.

DIAGNOSTICS — discusses the diagnostic indications that may be produced. Messages that are self explanatory are not listed.

WARNINGS — points out potential problems.

BUGS — gives known bugs and deficiencies.

2 Installation/Removal

Preparation

To add the AT&T 3B2 RISC System computer into an AT&T data switch network, you must install the host interface software and hardware into your computer, install hardware into the data switch, and make software configuration changes for the CPM-HS module in the data switch node.

This chapter shows you how to:

- Install the CommKit Host Interface hardware and software including TLI in your computer.
- Install and configure the CPM-HS module in the AT&T data switch node.
- Remove the CommKit TLI Support package from your computer.
- Remove the CommKit Host Interface software and hardware from your computer.

This chapter is organized so that you can move from section to section for the procedures required to install the CommKit Host Interface. Before adding any hardware or software:

1. Verify the correct UNIX System V Release is installed on your AT&T 3B2 RISC System computer that supports the current CommKit Host Interface release. See *Release Notes* for further details.
2. Have Datakit II VCS or BNS software loaded on your data switch node.
3. Have a small slotted screw driver, a phillips screw driver, and your computer's user guide available.
4. Have your data switch Maintenance Guide, Administrator's Guide, and Commands Reference Manual available.
5. Have the following AT&T HELP telephone numbers available:
   - For the CommKit Host Interface and AT&T data switch node or support contract information — **1-800-WE2-CARE***
   - For 3B2 Computer – **1-800-543-9935**

* To establish a service arrangement and receive support on these products, please contact your local SSC/CSC, your AT&T Sales Representative, or call the Data Networking Services Operations Group at 1-800-WE2-CARE.

Overview of Initial Installation

The procedures for initial installation of the CommKit Host Interface are as follows:

1. Install the CommKit Host Interface software and optional TLI Support package on your computer system.
2. Configure the CommKit Host Interface board.
3. Install the CommKit Host Interface board in the system.
4. Install the CPM-HS module in the AT&T data switch node.
5. Route the fiber optic cables from the computer system to the data switch node.
6. Connect the fiber optic cable.
7. Configure the CPM-HS module in the data switch node.
8. Run diagnostics on the CommKit Host Interface board.
9. Verify the operation of the hardware and software.
10. Customize the control tables.

Each of these procedures is described in detail in a major heading in this chapter.

Install the CommKit Host Interface Software

The CommKit Host Interface software package consists of the CommKit Host Interface software and a new version of the basic networking utilities (BNU).

Use the command ***pkginfo -l commkit*** to determine which version you may have already installed in your system. The PSTAMP field contains the version.

To load the software on your computer system, you must have a sufficient number of free blocks and inodes in the */etc/opt, /opt, root, /tmp, /usr*, and */var/opt* file systems. You can determine the amount of free space on your computer system by executing the ***df*** command which shows free inodes as free files.

Approximately 8000 blocks are required in the *root* file system to rebuild the UNIX System kernel and various configuration files. If */tmp* is a separate file system, it requires approximately 1000 blocks (which will be freed when the installation has been completed). Additionally, approximately 1000 blocks must be available in the */usr* and 8000 in the */var* file systems.

Pre-Installation Procedures

Before installing CommKit Host Interface software, you must do the following:

1. Review the Software Certificate for details on registering the copy-protected CommKit software. Obtain the CommKit Host Interface software key before proceeding with the installation.
2. Provide the area, exchange, and host name of your system. (This is optional and is used if you are customizing your server table. If you are, contact your data switch administrator for help with this information.) You will be prompted for this information if you wish to have customization done automatically to the control files used by **dkserver** (see *srvtab(4)*) during installation.
3. Remove any previously installed CommKit Host Interface software from your computer system. (Refer to the section *Remove the CommKit Host Interface Software* later in this chapter.)
4. Install the Networking Support Utilities (nsu) package before proceeding with the CommKit Host Interface software procedures.
5. Decide if you want to install the CommKit Host Interface manual pages. The manual pages take approximately 250 kbytes of disk space under the */usr* directory.

   Note: The manual pages are not required for system operation.

6. If your base UNIX System does not support the BNU over the AT&T data switch network, the software installation will ask if you want to load the new version of BNU.

Note: Unless you have installed your own version of BNU which you prefer to keep, we recommend you install the BNU version included with the CommKit Host Interface software.

Initial Installation Procedures

You can install the CommKit Host Interface Software by following the system prompts or, for a more streamlined approach, you can create a */etc/_dkhost_parms* file (described below) specifying all installation parameters. If you use this file, you will not be prompted during installation and removal.

Non-Prompted Mode (_dkhost_parms File)

If you create a file */etc/_dkhost_parms* to specify the value of the installation parameters, the installation / removal programs will not prompt for them. Table 2-1 lists the shell variables used by the installation / removal procedures.

Table 2-1: Installation/Removal Procedures – Shell Variables

Installation		
Variable Name	**Valid Value***	**Description**
CUSTOMIZE	y, n	Customize the server table
AREA, EXCH, HOST	user provided	Used for server table customization
MAN	y, n	Install online man pages
REMOVE_TABLE_I	y, n	Remove the current version of tables during package installation
UUCP	y, n	Install updated version of ***uucp*** commands
REMOVE_RC_I	y, n	Remove the */etc/init.d/dkitrc* file during package installation
Removal		
REMOVE_TABLE_R	y, n	Remove the current version of tables during package removal
REMOVE_RC_R	y, n	Remove the */etc/init.d/dkitrc* file during package removal

*The installation and removal software looks only for the value **y**; any other value, including null, will be interpreted as **n**, that is, do *not* remove, install, or customize. Any variable not included in the *_dkhost_parms* file will default to **n**.

Note: If any of the values AREA, EXCH, and HOST are missing, no customization will be done.

Using the *_dkhost_parms* file will eliminate Steps 2 through 8 under *Detailed Installation*. If this file is used, installation consists of the following:

1. Perform the steps under *General Installation*
2. Perform Steps 10 through 12 under *Detailed Installation*
3. Run the **dkregister** command to enter the Software Certificate number and Software Key after initial installation and before rebooting the system.

The installation and removal software will first validate the format of the */etc/_dkhost_parms* file. An example of a *_dkhost_parms* file is shown below:

```
# Install the manual pages
MAN=y
# Do not install the updated BNU programs
UUCP=n
# Do not remove current version of customized tables during installation
REMOVE_TABLE_I=n
# Do not remove current version of customized tables during removal
REMOVE_TABLE_R=n
# Do not remove /etc/init.d/dkitrc during installation
REMOVE_RC_I=n
# Do not remove /etc/init.d/dkitrc during removal
REMOVE_RC_R=n
# Customize the server table
CUSTOMIZE=y
AREA=myarea
EXCH=myexch
HOST=myhost
```

This is an example of a *_dkhost_parms* file that:

- Will install the manual pages online
- Will *not* install the updated ***uucp*** commands
- Will *not* remove the current version of the customized tables during package installation or removal

- Will *not* remove the current version of *dkitrc* in the */etc/init.d* directory during package installation or removal
- Will customize the server tables (using **myarea, myexch, myhost**).

The variable name must start in the first column; this must be followed by an equal sign; no spaces can be included before or after the equal sign. This must be followed, in turn, by the value of the variable. Comment lines (starting with #) are allowed. The installation / removal procedure will be terminated if the format or content is invalid. If the */etc/_dkhost_parms* file exists, the following message will appear:

```
The installation parameter file /etc/_dkhost_parms exists.
This installation will be run in NON-PROMPTED mode using the
values specified in this file.

Hit <DEL> key within the next 30 seconds if you do not wish to
install in NON-PROMPTED mode.
```

If you hit the **Delete** key, the following message will appear:

```
Installation will be run in PROMPTED mode.

Do you wish to continue the installation?
Enter y or n:
```

Enter **n** if you wish to examine the *_dkhost_parms* file.

General Installation

1. Log on as **root** on the console. Throughout the procedures that follow, system prompts and responses are shown in `constant width` typeface; user entries are shown in **`bold`** typeface.

   Note: Press the **Enter** key after every command you enter.

2. Insert the cartridge tape in the drive.
3. Enter the ***pkgadd*** command to load the CommKit Host Interface software package into your computer as shown in Figure 2-1, where:

   -d is an option of ***pkgadd*** for specifying the device option

   commmkit is the name of the package being installed.

4. For non-prompted installation, continue with Steps 2 and 3 under *Non-Prompted Mode.*
5. For prompted installation, go to the section *Detailed Installation*.

Detailed Installation

If you are not using the */etc/_dkhost_parms* file described previously, install the CommKit Host Interface Software as described below:

1. Follow the procedures under *General Installation*.

Figure 2-1: Install Software – Initial Dialogue Example

```
# pkgadd -d /dev/rSA/qtape1 commkit
Installation in progress. Do not remove the medium.
Processing package instance <commkit> from </dev/rSA/qtape1>
CommKit Host Interface to AT&T Data Switch
(u3b2v4) 4.4 v3
Copyright (c) 1995 AT&T
All rights Reserved
Do you wish to install the manual pages?
Enter y or n: y
Manual pages will be installed.
Saving file /usr/lib/libdk.so in /usr/lib/libdk.so_predk
```

2. If you want to install the manual pages, enter **y** as shown in Figure 2-1.

Note: The UNIX System is delivered with a BNU package which supports CommKit, therefore, only the *libdk.so* file will be saved (under *libdk.so_predk*) and restored to its original file name when the CommKit package is removed.

3. The system asks if you want to customize instances of Area / Exch and Area / Exch / Host in the first and last fields in the files under */etc/opt/dk/srvtab*.

```
The current server table, /etc/opt/dk/srvtab, contains entries
with 'Area', 'Exch', and 'Host'.  These occurrences
should be changed to the local names for your machine.
Would you like this customization to be done automatically
now? (You will be prompted to supply these local names.)
Enter y or n: y
```

4. If you want to customize, enter **y** and continue with the responses as shown on the following screen:

```
Enter Area: myarea
Enter Exch: myexch
Enter Host: myhost
Is this correct: myarea/myexch/myhost ? Enter y or n: y
## Processing package information.
## Processing system information.
   32 package pathnames are already properly installed.
## Verifying disk space requirements.

Installing CommKit Host Interface to AT&T Data Switch as <commkit>
```

```
## Installing part 1 of 1.
/dgn/DKPE
/dgn/X.DKPE
/etc/master.d/dkhs
/etc/master.d/dkmx
/etc/master.d/dkpe
/etc/master.d/dkty
/etc/master.d/dkux
/etc/master.d/dkx
/opt/dk/bin/dk
/opt/dk/bin/dkauth
/opt/dk/bin/dkcat
/opt/dk/bin/dkcu
                •
                •
                •
[ verifying class <man> ]
## Executing postinstall script.
```


5. Enter your area, exch, host when directed to do so in the screen above, and then **y** if the information is correct; **n** if it is not. You will be prompted for the information again if you enter **n**.
6. If you do *not* want to customize the files in */etc/opt/dk/srvtab* at this time enter **n** and the following screen will appear:

```
Reminder:  Don't forget to customize the /etc/opt/dk/srvtab after installation.
```

7. If you answer **y**, the system will respond with the following:

```
Customizing the files under /etc/opt/dk/srvtab...

All instances of 'Area/Exch' and 'Area/Exch/Host' in the first and
last fields in the files under /etc/opt/dk/srvtab have been changed to:
myarea/myexch and myarea/myexch/myhost respectively.
```

8. Enter the Software Certification Number and Software Key when prompted (see below). If necessary, you can defer entry during installation and manually run the **dkregister** command later when the values are available. Refer to the Software Certificate and *dkregister(1M)* manual page for details.


```
************************************
* CommKit Host Interface Registration *
************************************

You will now be prompted to enter your Software Certificate Number and
and Software Key.  These values must be properly entered to unlock the
copy protected CommKit Software.  Please refer to the Software
Certificate for details on obtaining a Software Key from the
WECARE Support Center (1-800-WE2-CARE).

You can continue without entering the values by typing "q" at
the appropriate prompt.  If you bypass entry of the Software
Certificate Number and Software Key at this time, you must
manually run the dkregister command before the CommKit Software
may be used.

Please enter your Software Certificate Number.
Certificate number (enter "q" to quit) > xxxxxx-xxxxxx-xxxx

You entered "XXXXXX-XXXXXX-XXXX".  Is this correct (y/n/q)? y

Please enter your Software Key.
Software Key (enter "q" to quit) > xxxxx-xxxx-xxxx-xxxx

You entered "XXXXX-XXXX-XXXX-XXXX".  Is this correct (y/n/q)? y
```

```
Registration Completed Successfully

Updating /etc/ttysrch
Updating edittbl

Installing dkpe module
Installing dkhs module
Installing dkux module
Installing dkty module
Installing dkx module
Installing dkmx module

Installation of <commkit> was successful.

*** IMPORTANT NOTICE ***
        If installation of all desired packages is complete
        the machine should be rebooted in order to
        ensure sane operation.  Execute the shutdown
        command with the appropriate options and wait for
        the "Console Login:" prompt.
```


9. The software installation is now complete.
10. Remove the cartridge tape.
11. If you will be installing TLI, proceed to the section *Install the CommKit TLI Package*, otherwise, enter the following commands to shut down the system so you can install the host interface board in your computer:

```
# cd /
# shutdown -y -g0 -i0
```

Upgrade Procedures

If you have a version of the CommKit Host Interface software installed on your computer and you want to install a later version, you must perform the following procedures.

1. Log on as **root** on the console.

   Note: Press the Enter key after every command you enter.

2. Remove the TLI package if it is installed (refer to the section *Remove the CommKit TLI Package* later in this chapter).
3. Remove the CommKit Software (refer to the section *Remove the CommKit Host Interface Software* later in this chapter).
4. Continue with Steps 2 and 3 under *General Installation*.
5. Continue with Steps 2 and 3 under *Detailed Installation*.
6. If you want to customize instances of Area/Exch and Area/Exch/Host in the first and last fields in the */etc/opt/dk/srvtab* files, enter **y** as shown in Step 4 under *Detailed Installation*.
7. If you saved the customized tables (**dkhosts**, **dkdotab**), and the service files under */etc/opt/dk/srvtab* directory during removal of the previous release of the CommKit Host Interface software, the following will be displayed:

```
Do you want to retain your current versions of files /etc/opt/dk/dkhosts,
/etc/opt/dk/dkdotab, /etc/opt/dk/dkgroups, and
all files under /etc/opt/dk/srvtab directory (y or n)?
```

If your *srvtab* file is a regular file, the following is displayed instead:

```
Do you want to retain your current versions of files /etc/opt/dk/dkhosts,
/etc/opt/dk/dkdotab, /etc/opt/dk/dkgroups, and
/etc/opt/dk/srvtab (y or n)?
```

8. If the saved files are different than the ones on the machine, the new version of control tables will be installed under the same name with the suffix **_4xvX**. This gives you the opportunity to compare the delivered files with your own version of the files. These files are */etc/opt/dk/dkdotab_4xvX*, */etc/opt/dk/dkhosts_4xvX*, */etc/opt/dk/dkgroups_4xvX*, and files with the suffix

_4xvX under */etc/opt/dk/srvtab* directory. If your */etc/opt/dk/srvtab* is a regular file, the delivered files will be installed in the */etc/opt/dk/srvtab_4xvX* directory.

Note: */etc/opt/dk/dkgroups* is introduced in Release 4.4v3; during an upgrade from an earlier version, the system will install a default *dkgroups* file.

Once you have finished comparing and making changes to your files, you should remove the suffix files (*_4xvX*) to insure a clean removal of the software if you ever wanted to remove the interface.

Note: The suffix **_4xvX** (where xvX represents the release and version number; e.g., **_44v2**) used for the delivered files will change for each new release or version of the CommKit Host Interface software.

Note: If a control table file has *not* changed, the system will not create a **_4xvX** file for that file. If you do not see a **_4xvX** version of a file, it means that the new control table file was identical to the old version which was, therefore, not saved.

9. If you saved the */etc/init.d/dkitrc* file during the removal of the previous release of CommKit Host Interface software, the following will be displayed:

```
Do you want to retain your current version of /etc/init.d/dkitrc (y or n)?
```

If you answer **y**, the file */etc/init.d/dkitrc* will not be overwritten and the delivered file will be installed as */etc/init.d/dkitrc_4xvX*. This allows you to compare your version with the delivered file.

10. The system will then respond with messages similar to the following:

```
## Processing package information.
## Processing system information.
   29 package pathnames are already properly installed.
## Verifying disk space requirements.

Installing CommKit Host Interface to AT&T Data Switch as <commkit>

## Installing part 1 of 1.
/dgn/DKPE
/dgn/X.DKPE
/etc/master.d/dkhs
/etc/master.d/dkmx
/etc/master.d/dkpe
/etc/master.d/dkty
/etc/master.d/dkux
/etc/master.d/dkx
/opt/dk/bin/dk
/opt/dk/bin/dkauth
/opt/dk/bin/dkcat
/opt/dk/bin/dkcu
                •
                •
                •
The following delivered control table files will be saved
   with suffix "_44v2":
         /etc/opt/dk/dkhosts               /etc/opt/dk/srvtab/do
         /etc/opt/dk/srvtab/pupu           /etc/opt/dk/srvtab/rx
         /etc/opt/dk/srvtab/whoami

Installing the delivered file /etc/init.d/dkitrc
   in /etc/init.d/dkitrc_44v2

Reminder: Don't forget to customize the /etc/opt/dk/srvtab after installation

/etc/opt/dk/dkuidtab will NOT be overwritten by the install procedure.
```

Note: The **dkuidtab** will be saved if you saved the tables during the removal process. Refer to the section *Remove the CommKit Host Interface Software* later in this chapter.

11. Enter **y** when asked to retain the Software Certificate and Key values:

```
**************************************
* CommKit Host Interface Registration *
**************************************

A valid CommKit registration record has been found on your system.
Do you wish to retain the existing Software Certificate Number
and Software Key values? (YyNn)? y
Updating /etc/ttysrch
Updating edittbl

Installing dkpe module
Installing dkhs module
Installing dkux module
Installing dkty module
Installing dkx module
Installing dkmx module

Installation of <commkit> was successful.

*** IMPORTANT NOTICE ***
        If installation of all desired packages is complete
        the machine should be rebooted in order to
        ensure sane operation.  Execute the shutdown
        command with the appropriate options and wait for
        the "Console Login:" prompt.
```


12. The software installation is now complete.
13. Remove the cartridge tape.
14. If you have more than one board installed or are not using the default number of channels (64) for each interface, you must customize the script file, */etc/init.d/dkitrc*. Refer to the section *dkitrc Script File* in Chapter 4.
16. If you are going to install TLI, proceed to the next section, otherwise, enter the following command to reboot the system:

```
# shutdown -y -i6 -g0
```

Install the CommKit TLI Package

This package is optional. If you do not wish to install it, go to the next section. The CommKit Host Interface software must be installed before the TLI package. Refer to *Using the TLI Package* in Chapter 4 and the *dktli(7)* manual page.

To install the TLI Package:

1. Log on as **root**.
2. Insert a cartridge tape in the drive.
3. Enter the ***pkgadd*** command to load the TLI package:

```
# pkgadd -d /dev/rSA/qtape1 cktli
```

-d is an option of ***pkgadd*** for specifying the device option

cktli is the name of the package being installed.

4. Once the software has been installed, remove the tape.
5. If you are going to install the host interface hardware, shut down the system and proceed to the next section:

```
#  cd /
# shutdown -y -i0 -g0
```

6. If the host interface hardware is already installed, reboot the system:

```
#  cd /
# shutdown -y -i6 -g0
```

Install Host Interface Hardware

The CommKit Host Interface Software must be installed before the host interface board can be installed. Read this entire section before installing the board.

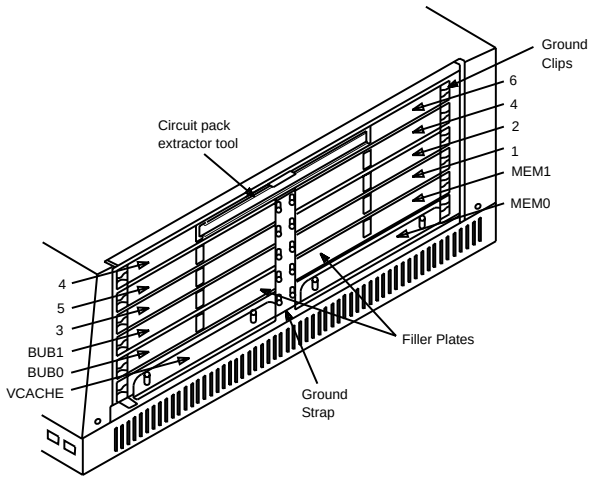
Procedure

If the 3B2 computer is running, turn it off as described in the *3B2 Owner/Operator Manual* and unplug the power cord. Install the host interface board:

Note: The host interface board must be installed in the next available connector slot with two vertical slots open. You cannot skip a backplane slot.

Note: You may have to rearrange previously installed boards before installing the host interface board. If so, refer to the documentation provided with the boards you will be moving.

Figure 2-2: 3B2/500 Cabinet Backplane Slots

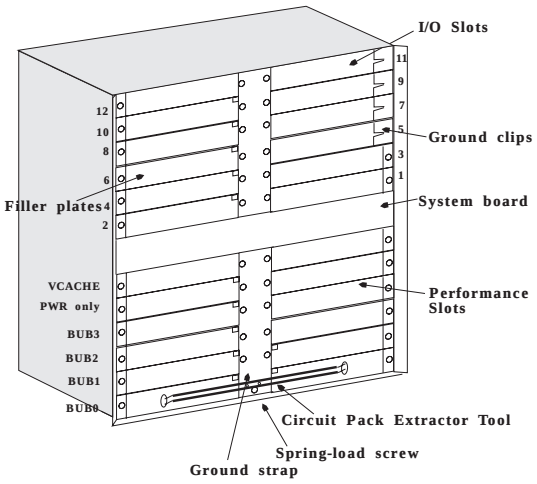


1. Refer to Figures 2-2 and 2-3 for typical 3B2 equipment.
2. Remove the ground strap by removing the screws holding it to the filler plate and host interface board. Push and turn the spring-loaded screw a quarter turn as shown in Figures 2-2 and 2-3.

Caution: Do not install the host interface board into a performance slot on the 3B2/500 or 3B2/600. Doing so may damage both the 3B2 and the host interface board.

Caution: Handle the host interface board by the edges only.

Figure 2-3: 3B2/600 Cabinet Backplane Slots



3. Remove the appropriate filler plates to install the host interface board.
4. If the filler plate covers two backplane slots and you need only one of them:
   a. Break the filler plate in half

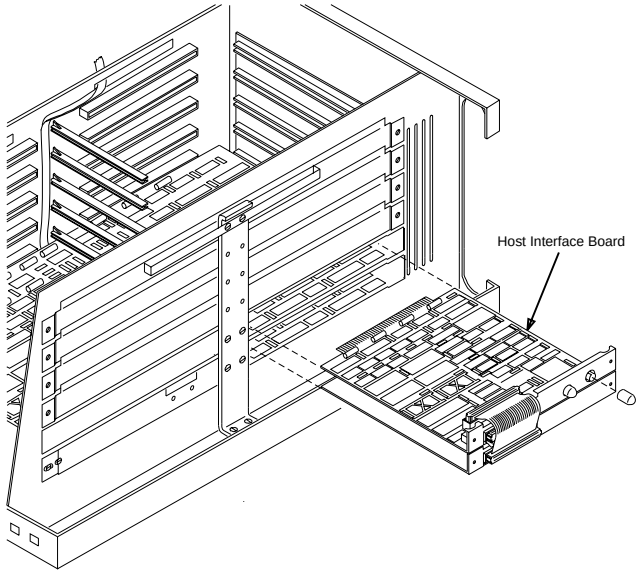
b. Install half over the slot not being used

c. Save the other half for possible future use.

Warning: Failure to replace the filler plates over the unused slots will disrupt airflow and may cause radiation noncompliance with Class B limits of Subpart J, Part 15 of FCC rules.

5. Insert the host interface board into the selected slot with the component side up as shown in Figure 2-4.

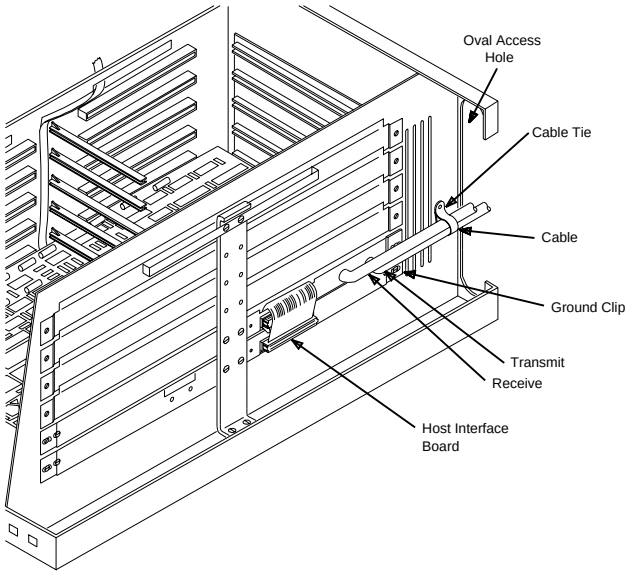
Figure 2-4: Inserting the Host Interface Board



6. Replace the ground strap and install the ground clip as shown in Figure 2-5.

Note: Install the ground clip at the outside end of the faceplate opposite the ground strap.

Figure 2-5: Host Interface Board Installed



Connect the fiber optic cable as described in the section *Connect the Fiber Optic Cable* later in this chapter.

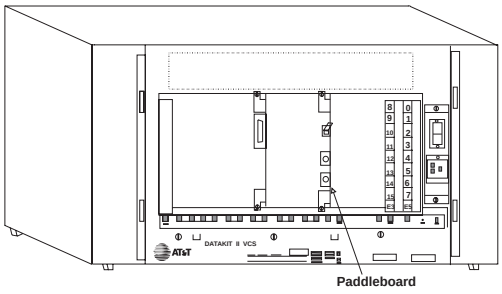
Install the CPM-HS Module in the Data Switch Node

The CPM-HS module will be installed in the data switch node. Refer to the appropriate AT&T Installation and Maintenance Guides (see Chapter 1, Table 1-3) for additional details.

To install the CPM-HS in the node:

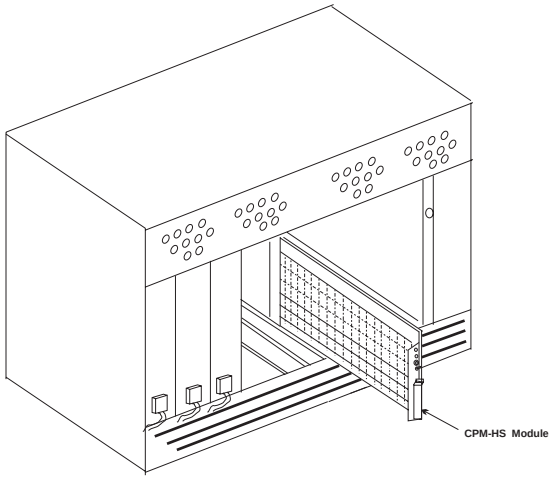
1. Face the data switch cabinet from the rear. Slide the paddle board into one of the available backplane slots. (See Figure 2-6).

Figure 2-6: Typical Rear View of a Data Switch Cabinet



2. Connect the paddle board to the chassis.
3. Face the cabinet from the front. Slide the CPM-HS module into the same number slot as the paddle board. (See Figure 2-7).

Figure 2-7: Typical Front View of a Data Switch Cabinet



4. Enable the CPM-HS module by moving the module faceplate switch to the ENABLE position.

Route the Fiber Optic Cable

General

The fiber optic cable may be run in overhead ceilings, in subfloor cable runs, and in riser shafts. Before routing the fiber optic cable, you must connect rubber caps to each end of the cable. This will protect the fiber optic cable from dirt or dust during installation.

When routing the fiber optic cable, you should keep the fiber optic cable away from copper riser cables. If you are unable to do so, install an inner liner (conduit) to keep the cables separated.

Carlon EFT corrugated tubing (or equivalent) may be used to separate the fiber optic cable from copper riser cables. This corrugated tubing can be used in short lengths and can be formed into bends.

Caution: Fiber optic cables are not intended for use in air handling ceiling areas unless installed in approved conduit.

When installing the fiber optic cable, you should avoid tight pulls or tugs against sharp corners of framework. If fiber optic cables are to be installed around sharp edges of cabinetry or framework, cover the edges with split tubing or similar material. When lacing or securing the fiber optic cable, use flat lacing twine or cable ties; do not tie the fiber optic cable too tight because microbending losses may occur. Bundles of cables should not hang or protrude into the work space. Wrap the cables into loops not less than three inches in diameter, although short-term handling into loops of one-inch diameter is satisfactory.

Tools and Hardware

Tools and hardware (such as fish wire, woven cable grips, or rope) used to install copper wire and cable in building duct and conduit systems are satisfactory for use in installing fiber optic cable. If woven cable grips are used with fiber optic cables, tape them to the cable jacket before pulling the cable.

Conduit Installation

The fiber optic cables were not designed for conduit installation, but they may be installed in a conduit if the following applies:

1. Only two fiber optic cables into a single conduit
2. The pull force of the fiber optic cable does not exceed 50 pounds per cable.

Fiber optic cables should not be pulled through more than four 90-degree bends. If the conduit run contains more than four 90-degree bends, provide intermediate *help* points. The minimum recommended conduit bend radius is 4½ inches.

Caution: Never pull the cable around a sharp corner, such as a junction box connection.

Warning: Do not install fiber optic cable in conduits with less than ¾-inch inside diameter.

Pulling tension during conduit installation can be minimized by the following:

1. The fiber optic cable should enter the end of the conduit nearest the curved sections.
2. Ducts or conduits should be free of foreign obstructions before cable installation.
3. The following lubricants are recommended for PVC cabling:
   - — Polywater A&C — American Polywater Corp.
   - — Hydralube Blue — Arnco Equipment Co.
   - — Neutral soft soap
   - — Talcum powder.

Note: Do not use a petroleum-based lubricant on PVC cables.

Connect the Fiber Optic Cable

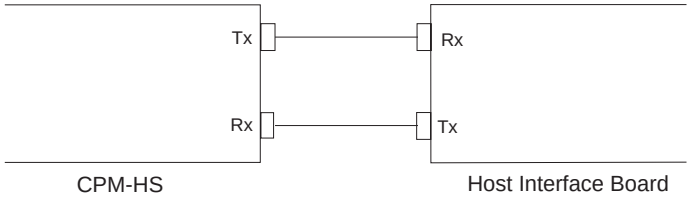
The length of the fiber optic cable between the node (CPM-HS module) and the computer system (CommKit Host Interface board) should not exceed 1 kilometer.

Connect the fiber optic cable as follows (refer to Figure 2-8):

1. Face the data switch cabinet from the rear. The receiver (Rx) is the connector at the top and the transmitter (Tx) is the bottom connector.

2. Remove the rubber caps from the fiber optic cable and the Transmitter and Receiver connectors of the Host Interface Board. Save the caps for later use.
3. Connect the Transmitter side of the Host Interface board in the computer to the Rx side of the CPM-HS module in the data switch node.
4. Connect the Receiver side of the Host Interface board in the computer to the Tx side of the CPM-HS module in the data switch node.
5. Press the ends of the cable tie together. The holes in the ends of the tie should line up with the hole near the rear of the cabinet.
6. Place the screw through the tie and into the hole near the rear of the cabinet. Tighten the screw to support the cabinet.

Figure 2-8: Fiber Cable Connections



Configure the Data Switch Control Computer Database

Once the CPM-HS module is plugged into the data switch node, the CPM-HS module must be configured from the data switch node console. Refer to the appropriate AT&T Commands Reference documentation (Chapter 1, Table 1-3) for more information.

To configure the node, you must:

1. Define the group name
2. Define the address* for the **dkserver** service
3. Define the address for the ***listener***
4. Enter the CPM-HS module.

To accomplish these steps, conduct the dialogues shown below.

Note: Dialogues differ depending on the AT&T data switch and release number. The figures show typical dialogues for R2.0 of the Datakit II VCS and BNS data switches. Significant differences for Datakit II VCS R1.0 are described as necessary.

Define Group Name

Conduct the dialogue as shown below (responses based on Figure 1-2):

```
CC0> enter group
GROUP [up to 8 chars]: bird
TYPE [local, trunk: +(local)]: local
DIRECTION [originate, receive, 2way]: 2way
DEVICE OR HOST [up to 8 chars]: bird
PASSWORD [up to 8 chars, none: +(none)]:none
ROUND ROBIN SERVICE [per_port, per_module, none: +(none)]: none
GROUP [up to 8 chars]:  Delete
CC0>
```

The entries are described below:

group Defines the name of the computer system as known by the data switch. This is **bird** in the example shown in the screen above. (Enter **Delete** to end the session.)

* The term *address* in the BNS and R2.0 of the Datakit II VCS data switch is the same as the term *name* in R1.0 of the Datakit II VCS data switch. When using the ***enter address*** command to define the name, select **mnemonic** as the type. Refer to the examples that follow.

type	The computer system is a local connection to the data switch. Enter **local**.
direction	The fiber interface may be used to originate and receive calls through the CPM-HS module, therefore, it is **2way**.
device or host	We recommend assigning the same name as the name of the computer system.
password	Passwords are not used with the CPM-HS module.
round robin	We do *not* recommend using **per_port**. Enter either **per_module** (for multiple boards) or **none**.

Define Address for dkserver Service

The server name used by **dkserver** must match an address configured for the CPM-HS or host. (The following example is based on the example shown in Chapter 1, Figure 1-2 and uses the default **dkserver** name.) Conduct the dialogue as shown in the screen below:

```
CC0> enter address
LEVEL [local, area, exchange, local, speedcall: +(local)]:  local
TYPE [x121, mnemonic, both: +(mnemonic)]: mnemonic
MNEMONIC ADDRESS [up to 8 chars]: bird
PAD SUPPORT [yes, no: +(no)]:  Enter
DIRECTORY ENTRY [up to 30 chars double quoted, none: +(none)]:
"name of the default dkserver"
GROUP(S) [up to 4 groups separated by commas, none: +(none)]:
bird
ORIGINATING GROUP NAME SECURITY PATTERN(S)
[comma-separated pattern list, same_as, none: +(none)]:  Enter
INITIAL SERVICE STATE [in, out: +(out)]: in
LEVEL [network, area, exchange, local, speedcall: +(local)]:  Delete
CC0>
```

Note: The command ***enter address*** in the BNS or R2.0 of the Datakit II VCS data switch is analogous to the ***enter name*** command in R1.0 of the Datakit II VCS.

level	Enter **local** (R2.0) or Delete to end the session.
type	The addressing type is **mnemonic** (R2.0 and later) or **standard** (R1.0).
mnemonic address/name	This is the local service address (name R1.0) used by the computer system and it must be the same as the server name that is assigned to the **dkserver**.
PAD	This is not used (R2.0).
directory entry	This is a 30-character description of the **dkserver** enclosed in double quotes.
group	Enter the name of the data switch group used to direct calls to your computer system.
security pattern	The originating group name security feature allows the data switch to restrict calls for this host to a select group of data switch originating groups. Since the *srvtab* file may also be used to restrict incoming calls, Enter is used to disable this feature.
service state	The local service address must be *in service* for the computer system to originate and receive calls (R2.0).
restore to service	The local service address must be *in service* for the computer system to originate and receive calls (R1.0).

If you do *not* intend to use TLI support, go to the section, *Configure the CPM-HS Module*, later in this chapter. If TLI Support is to be used, continue with the next section, *Define Address for the Listener*.

Define Address for the Listener

The server name used by the ***listener*** must match an address configured for the CPM-HS or host. In general, the server name for the ***listener*** will be the upper case version of the host name (***uname***). For example (in Figure 1-2), host = **bird**; server name for listener = **BIRD**. Conduct the dialogue as shown in the following screen:

```
CC0> enter address
LEVEL [local, area, exchange, local, speedcall: +(local)]:  local
TYPE [x121, mnemonic, both: +(mnemonic)]: mnemonic
MNEMONIC ADDRESS [up to 8 chars]: BIRD
PAD SUPPORT [yes, no: +(no)]:  Enter
DIRECTORY ENTRY [up to 30 chars double quoted, none: +(none)]:
"name of the listener"
GROUP(S) [up to 4 groups separated by commas, none: +(none)]:
bird
ORIGINATING GROUP NAME SECURITY PATTERN(S)
[comma-separated pattern list, same_as, none: +(none)]:  Enter

INITIAL SERVICE STATE [in, out: +(out)]: in

LEVEL [network, area, exchange, local, speedcall: +(local)]:  Delete
CC0>
```


level — Enter **local** (R2.0) or **Delete** to end the session.

type — The addressing type is **mnemonic** (or **standard** for Datakit II VCS R1.0 only).

mnemonic address/name — Enter the local server address (name R1.0) used by the listener. This is **GULL** in this example.

PAD — This is not used (R2.0).

directory entry — This is a 30-character description of the listener enclosed in double quotes.

group — Enter the name of the data switch group that is used to direct calls to your computer system.

security pattern — The originating group-name security feature allows the data switch to restrict calls for this host to a select group of data switch originating groups. Since the *srvtab* file may also be used to restrict incoming calls, **Enter** is used to disable this feature.

service state — The local service address must be *in service* for the computer system to originate and receive calls (R2.0).

restore to service — The local service address must be *in service* for the computer system to originate and receive calls (R1.0).

Configure the CPM-HS Module

Conduct the dialogue as shown in the screen below:

```
CC0> enter cpm
MODULE ADDRESS [2-127]: 6
COMMENT [up to 60 chars double quoted]:
"fiber interface"
HARDWARE TYPE [422, hs: +(hs)]: hs
NUMBER OF CHANNELS [2-512: +(32)]:64
SINGLE OR MULTIPLE GROUP(S) [single, multiple: +(single)]: multiple
GROUP [up to 8 chars]: bird
CHANNEL RANGE [low-high: (+2-63)]: 2-30
ENDPOINT NUMBER OR RANGE [0000-9999, none: +(none)]: none
105 Channels Left to Be Specified
GROUP [up to 8 chars]: bear
CHANNEL RANGE [low-high]: 31-63
ENDPOINT NUMBER OR RANGE [0000-9999, none: +(none)]: none
MODULE ADDRESS:  Return
CC0> restore cpm 6
CC0>
```

address — Check the data switch cabinet for the slot number of the CPM-HS module to which the host is connected. Enter that slot number; in this example the CPM-HS module is in slot 6.

comment — A description of the entry.

hardware type — Host's fiber interface hardware always connects to the CPM-HS.

number of channels — The default number of channels for the CommKit Host Interface software is 64. Enter the same number as the number of channels configured on your computer system. To change the number of channels, refer to Chapter 4.

Single or multiple — Enter **single**, or **multiple** if channel groups are used.

group — Enter the name of the data switch node channel group being used to direct calls to your computer system. Enter additional channel groups as required by your configuration (refer to the section *dkgroups* in Chapter 3).

Endpoint Not used (R2.0).

Note: The CPM-HS module must be restored to service before use.

Run Diagnostics on Host Interface Module

The host interface module is tested by running off-line diagnostics. Use these diagnostics to locate hardware problems in the 3B2 computer.

Caution: Diagnostic phases should only be run by an experienced user. If you have a system failure and are not confident about running diagnostics, call your AT&T Service Representative or authorized dealer.

Diagnostics are run from the 3B2 firmware mode. You must be logged in as **root** on the 3B2 to bring the system to the firmware mode. To run the diagnostics:

1. Go to the firmware mode by conducting the following dialog:

```
# /etc/shutdown -y -i5
FIRMWARE MODE xxxxxx
Enter name of program to execute [/etc/system]: dgmon
```

Note: Enter your password (shown as **xxxxxx** above) when the FIRMWARE MODE prompt appears. The password will not be echoed onscreen.

2. Boot the system by executing the **dgmon** program from the hard disk. Refer to the *3B2 RISC Systems UNIX System V R4.0.3 Owner/Operator Manual*.

3. Run the diagnostics by entering:

```
DGMON> dgn dkpe ph=1-15
DGMON>
```


Note: For phases 13–15 to pass, the fiber optic cable must be connected to the data switch node.

4. Quit the diagnostics mode by entering **q** at the DGMON> prompt above.

If a diagnostic failure occurs, verify that the fiber optic cable is properly connected between the 3B2 Computer and the data switch node and that the CPM-HS module is properly inserted into the correct data switch slot and that it is in service. If you continue to experience diagnostic failures, refer to the section *Customer Assistance* in Chapter 1.

Diagnostic Phases

Table 2-2: Diagnostic Phases

Phase	Type	Description	Phase	Type	Description
1	Demand	CIO Sanity Test	9	Demand	Programmed I/O Byte
2	Demand	PCSR Write/Read Test	10	Demand	Programmed I/O Word
3	Demand	Upper RAM Write/Read Test	11	Demand	DMA Transfer Byte Test
4	Demand	Lower RAM Write/Read Test	12	Demand	DMA Transfer Word Test
5	Demand	CPU Chip Select Test	13	Normal	FIB Internal Loop
6	Demand	CPU DMA Internal Test	14	Demand	FIB External Loop
7	Demand	CPU Timer Test	15	Normal	DMA FIB External Loop
8	Demand	CPU Interrupt Controller			

Table 2-2 shows the diagnostic phase descriptions for 3B2 RISC computer. The following screen shows a typical output for the diagnostic phases.

```
DGMON> dgn dkpe ph=1-15
     <<< DIAGNOSTIC MODE >>>

DKIT Phase: 1  Test: Common I/O Sanity   Type: DEMAND
Time Taken = 1 second
*** DKIT  Phase: 1  Diagnostic PASSED ***

DKIT Phase: 2  Test: Control and Status Register   Type: DEMAND
Time Taken = 1 second
*** DKIT  Phase: 2  Diagnostic PASSED ***

DKIT Phase: 3  Test: Upper RAM   Type: DEMAND
Time Taken = 2.5 minutes
*** DKIT  Phase: 3  Diagnostic PASSED ***

DKIT Phase: 4  Test: Lower RAM   Type: DEMAND
Time Taken = 2.5 minutes
*** DKIT  Phase: 4  Diagnostic PASSED ***

DKIT Phase: 5  Test: Central Processor Unit   Type: DEMAND
Time Taken = 1 second
*** DKIT  Phase: 5  Diagnostic PASSED ***

DKIT Phase: 6  Test: Central Processor Unit   Type: DEMAND
Time Taken = 1 second
*** DKIT  Phase: 6  Diagnostic PASSED ***

DKIT Phase: 7  Test: Central Processor Unit   Type: DEMAND
Time Taken = 1 second
*** DKIT  Phase: 7  Diagnostic PASSED ***

DKIT Phase: 10  Test: Central Processor Unit   Type: DEMAND
Time Taken = 1 second
*** DKIT  Phase: 10  Diagnostic PASSED ***

DKIT Phase: 9  Test: BYTE Programmed I/O   Type: DEMAND
Time Taken = 1 minute
*** DKIT  Phase: 9  Diagnostic PASSED ***

DKIT Phase: 10  Test: WORD Programmed I/O   Type: DEMAND
Time Taken = 1 minute
*** DKIT  Phase: 10  Diagnostic PASSED ***

DKIT Phase: 11  Test: BYTE DMA Transfer   Type: DEMAND
Time Taken = 1 second
*** DKIT  Phase: 11  Diagnostic PASSED ***

DKIT Phase: 12  Test: WORD DMA Transfer   Type: DEMAND
Time Taken = 1 second
*** DKIT  Phase: 12  Diagnostic PASSED ***

DKIT Phase: 13  Test: FIB Internal Loop Around   Type: NORMAL
Time Taken = 3 seconds
WARNING - THE FIBER OPTIC CABLE MUST BE CONNECTED!
```

```
*** DKIT  Phase: 13  Diagnostic PASSED ***

DKIT Phase: 14  Test: FIB External Loop Around   Type: DEMAND
Time Taken = 3 seconds
WARNING - THE FIBER OPTIC CABLE MUST BE CONNECTED!
*** DKIT  Phase: 14  Diagnostic PASSED ***

DKIT Phase: 15  Test: DMA - FIB External Loop Around   Type: NORMAL
Time Taken = 5 seconds
WARNING - THE FIBER OPTIC CABLE MUST BE CONNECTED!!
*** DKIT  Phase: 15  Diagnostic PASSED ***

      DKIT 0 (IN I/O BUS SLOT 2) DIAGNOSTICS PASSED
```


Verify Operation

Now that you have installed the CommKit Host Interface software and hardware into your computer and installed the hardware in the AT&T data switch node and configured the node, you are ready to verify that your computer can communicate successfully with that node. You must be logged in as **root** on your computer, and the system must be in the **multi-user** mode (init state 2). The following is an overview of these step-by-step procedures:

1. Verify the operation of the **dkdaemon** process.
2. Verify the operation of the **dkserver** process.
3. Verify data transfer across the interface.
4. Verify terminal login across the interface.
5. Verify operation of the remote login facility.

Verify the Operation of the dkdaemon Process

To start the newly-installed interface, the interface **daemon** process must be started. The interface **daemon** (*/opt/dk/sbin/dkdaemon*) is started automatically whenever the system enters init state 2. See the manual page *dkitrc(1M)* for more details on what processes are started automatically at init state 2.

If you have more than one CommKit Host Interface board installed, you can start separate **daemon** processes for each interface by using the following command where *n* is the interface board number:

```
# /opt/dk/sbin/dkdaemon -in
```

If the **-i** option is not used with the **dkdaemon** command, a single daemon process services all interfaces. If the number of channels that you wish to configure is not the same for each interface, you must have separate **daemon** processes for each interface.

1. Check to see if the **daemon** process is running by using the ***ps*** command. If the process is not running, manually start the **dkdaemon** by entering the following where *nChannels* is the number of channels that you wish to configure on your interface [*dkdaemon(1M)*].

```
# /opt/dk/sbin/dkdaemon [-c nChannels] -in
```

Note: If you omit the *-c nChannels* option, the daemon will default to 64 channels. In either case, the number of channels must be the same as the number configured on the CPM-HS module or incoming and outgoing call attempts may fail.

2. Verify that the **daemon** is running, look in the log file by entering the following command:

```
# tail /var/opt/dk/log/dkdaemonlog
```

The system will respond with a series of log entries indicating the interface is starting or ACTIVE. Refer to the *dkdaemon (1M)* manual page for information regarding the log entries.

3. After you start the **daemon** successfully, a message similar to the following will be displayed on the data switch console where *XX* is the slot number of the CPM-HS module that connects your computer to the data switch node.

```
<date><time> NODE=<nodename>
8620 MODADR=XX  MODTYPE=cpmhs
REPORT STATUS: unixcsc: Host is active
```

Verify the Operation of the dkserver Process

To test the newly-installed interfaces, a server process must be started for each hardware interface. This server (*/opt/dk/sbin/dkserver*) is started automatically whenever the system enters init state 2. See the manual page *dkitrc(1M)* for more details.

Note: Each hardware interface that is to receive calls from the AT&T data network switch requires its own server process.

Check to see if the server process is running by using the ***ps*** command. If the server process is not running, manually start the server by performing the following steps:

1. At the data switch, verify that the CPM-HS is in service by using the ***verify*** command shown below (for a CPM-HS module in slot 8):

```
CCO> verify cpm 8
        93-04-07 16:06:43  NODE=ocean
M       verify cpm 8
        MODULE ADDRESS: 8
        MODULE TYPES: cpmhs       NCHLS: 64
        SERVICE STATE: in SERVICE TYPE: unix
        HOST: plover
COMMENT:

BILLING          PERIODIC BILLING
off              off

GROUP    CHNLS    CHNL RANGE       EPN      CUG PROFILE
plover    62       2-63
```


2. If the output of the ***verify*** command shows that the CPM-HS module is out of service, the module must be restored to service by entering the following command on the data switch node console where *XX* is the number of the slot to which the CPM-HS module is located in the data switch node:

```
CCO> restore cpm XX
```

3. Verify that the server name to be used by **dkserver** is in service by using the ***verify*** command shown below [for a server named **plover** (on R1.0 ***verify name plover***)]:

```
CCO>verify add local plover
       88-04-20   14:30:55   NODE=ocean
M      verify address local plover

       MNEMONIC ADDRESS: plover    X.121  ADDRESS:
       LEVEL: local      SERVICE STATE: in

PAD SUPPORT:  no

DIRECTORY:  none
SECURITY:  one

GROUP:  plover

CCO>
```

4. If the output of the verify command shows the name is out of service, the name must be restored to service by entering the following command on the data switch node console where **servername** is the name that is used by the CommKit Host Interface server:

```
CC>restore addr local servername
```

5. On the UNIX system, start the server. The following is an example of starting a server, **plover** for hardware interface 1:

```
#  /opt/dk/sbin/dkserver -i 1 -s plover -v 9
```

1 is the number of the CommKit Host Interface board for which the server is being started.

plover is the name of your computer as known by the data switch node.

9 is the level at which the server activity is logged.

For more information, refer to manual pages *dkserver(1M), srvtab(4), dkuidtab(4),* and *dksrvlog(4).*

This server will use the default *controlfile* and *uidfile* (*/etc/opt/dk/srvtab* and */etc/opt/dk/dkuidtab*). All logged server activity will be written to the default log-file */var/opt/dk/log/dksrvlog*.

To verify that the server has been started, examine the last few entries in the server log (default is */var/opt/dk/log/dksrvlog*). A successfully started server will log the following messages:

```
date (pid) [x.yyy] SERVER name is INITING files=(srvtab uidtab) loglvl=n
date (pid) [x.yyy] dkmgr: SERVER name is ACTIVE and SERVING
dkmgr: chans per intfc=c window size=m protocol=p
```

Note: The third line above only appears if **dkserver** was started with a **-v** option value of **8** or more.

pid is the process id of the dkserver process.

x is the interface number.

yyy is the channel number that the server is running on.

name is the name of the started server.

srvtab is the server table.

uidtab is the authorization table.

n is the level at which the server activity is logged.

c is the number of channels used when the **dkdaemon** process was started. If this value does not equal the number of channels per interface as defined in the data switch control computer database, then the daemon was started with the wrong number of channels or the entry for the CPM-HS was configured incorrectly. Either the value in your computer or the value in the data switch control computer database must be changed so that the number of channels are the same.

To change the number of channels on your computer, refer to the section

Changing the Number of Channels in Chapter 4.

m is the receive window size.

p is the call processing protocol version.

After verifying that the server has started, verify that it is actually communicating with the data switch node. Enter the following command on the data switch console to display a summary of the virtual circuits that are active on the CommKit Host Interface:

```
CC> display connections mod XX
```

XX is the number of the CPM-HS slot. The output of the ***display*** command should verify that at least one channel is in the SERVING state. If it is not, then the server (*/opt/dk/sbin/dkserver*) is not successfully communicating with the data switch node; refer to the section *Troubleshooting the Host Interface Communication* in Chapter 4.

The following is an example of the ***display*** command for a CPM-HS module in slot 8:

```
CC> display connections mod 8

   90-11-13 13:53:04 NODE=ocean
M  display connections mod 8
MODULE:   8
-------------
   CH/PT  CU/TM GROUP     PKT CNT  STATE     MOD ADD  CH/PT CU/TM GROUP    PKT CNT
          BOARD                    (+ = PDD)               BOARD
    1           ****         4923  ACTIVE
    2           plover         40  SERVING

CC>
```

Verify Data Transfer Across the Interface

The command **/opt/dk/sbin/dkload** [see *dkload*(1M)] can be used to perform a loop-around data transfer to load test and check for possible errors and / or problems on the CommKit Host Interface.

/opt/dk/sbin/dkload sets up a virtual circuit between one or two CommKit Host Interface hosts and then sends and receives data across the data switch network circuit.

The format of this command is as follows:

```
# /opt/dk/sbin/dkload dialstring.dkload [-s messagesize] \
[-n number of iterations] [-l logfile]
```

To run a loop-around load test, the **dkload** command should use the local machine dialstring that connects the local machine to itself.

In addition, there must be a **dkload** table in the */etc/opt/dk/srvtab* directory on the destination host. The default system has this table already included.

For example:

```
# /opt/dk/sbin/dkload dialstring.dkload -s 512 -n 1000
```

dialstring is the name of the host as known by the data switch node. This will result in 1000 messages of size 512 bytes each to be sent loop-around to and from the host dialstring and the results of the test sent to the file */tmp/dkloadlog*. To verify that the tests have passed, tail the file */tmp/dkloadlog*. There should be entries in the log file similar to the following:

```
PID: LOCAL TEST FINISHED OK at DATE
        nwritten=512, sendsize=512, countsent=1000
PID: REMOTE TEST FINISHED at DATE
PID: REMOTE TEST FINISHED OK nread=512, countread=1000
```


PID	is the process id of the outgoing **dkload** process (LOCAL) or the process id of the incoming **dkload** process (REMOTE).
DATE	is the date and time the test completed.
nwritten	is the return value from the final *write* system call.
sendsize	is the number of bytes sent for each iteration.
countsent	is the number of iterations actually sent.
nread	is the return value from the final *read* system call.
countread	is the number of iterations actually received.

The values of **countsent** and **countread** should be equal to the number specified on the **/opt/dk/sbin/dkload** command line for the *-n* flag. Any other form of entry indicates a failure. The values of **nwritten** and **nread** should be equal to the number specified on the **/opt/dk/sbin/dkload** command line for the *-s* flag.

In the case of a failure:

1. Examine the *dkload* log (*/tmp/dkloadlog* in the command line shown previously) for an indication of the reason for the failure. See *dkload(1M)* for a description of the error encountered.
2. Refer to the section entitled *Software Troubleshooting Procedure* in Chapter 4.
3. Finally, remove the log file */tmp/dkloadlog*.

Verify Terminal Login Across the Interface

After verifying that the server is talking to the CommKit Host Interface, we suggest you test the operation of the terminal (tty) interface as follows:

1. Verify that there is a terminal connected to the data switch network that has access to your computer via the CommKit Host Interface.
2. Enter the following where *area/exchange/host* is the full address of the server name used by your computer as entered in the data switch control computer database:

```
DESTINATION:area/exchange/host
login:
```


3. Continue the terminal session as a normal login session.
4. In the event that the login sequence fails, refer to the section *Software Troubleshooting Procedure* in Chapter 4.

Verify Operation of the Remote Login Facility

We suggest you test the operation of the remote login facility as follows:

1. Enter the following command from a terminal logged on to the host where *area/exchange/host* is the address of the server name being used by your computer as entered in the data switch control computer database.

```
# su listen -c "dk area/exchange/host"
login:
```

   The ***su*** command is used to change user UID to the **listen** user ID because the default **/etc/opt/dk/srvtab** server table traps and rejects remote login calls that attempt to use the **root** login.

2. Continue the session as a normal login session.
3. In the event that the login sequence fails, refer to the section *Software Troubleshooting Procedure* in Chapter 4.

Customize the Control Tables and dkitrc

The CommKit Host Interface software package relies on a number of files (tables) which must be edited to reflect your own community of interest. These files control external access to your host and help direct requests to other remote host.

Note: The software installation process is not complete until these files have been edited.

When using customized table files, the new version of control tables will be installed under the same name as the old files with the suffix **_4xvX**, if the saved files are different than the files on the machine.

Note: The system will always save the file */etc/init.d/dkitrc_4xvX*, even when the delivered and customized versions are the same.

This gives you the opportunity to compare the delivered files with your own version of the files. (These files are */etc/opt/dk/dkdotab_4xvX*, */etc/opt/dk/dkhosts_4xvX*, */etc/opt/dk/dkgroups_4xvX*, and files with the suffix _4xvX under */etc/opt/dk/srvtab* directory. If your */etc/opt/dk/srvtab* is a regular file, the delivered files will be installed in the */etc/opt/dk/srvtab_4xvX* directory.)

Note: The suffix **_4xvX** (where xvX represents the release and version number; e.g., **_42v3**) used for the delivered files will change for each new release or version of the CommKit Host Interface software.

Note: If a control table file has *not* changed, the system will not create a **_4xvX** file for that file. If you do not see a **_4xvX** version of a file, it means that the new control table file was identical to the old version which was, therefore, not saved.

The files are:

/etc/opt/dk/srvtab directory	files under this directory must be edited to reflect local area, exchange, and host names and to grant external access to your computer through the data switch node
/etc/opt/dk/dkhosts	to be edited to reflect local host names
/etc/opt/dk/dkdotab	to be edited to reflect local services.
/etc/opt/dk/dkgroups	to be edited to reflect local channel groups.

Another file which may be customized is the shell script */etc/init.d/dkitrc.* This script has links in several of the */etc/rc?.d* directories and is executed each time an init state transition occurs. Lines for starting additional **daemons** or **servers** may be added to this shell script. Refer to the section *Administration Notes* in Chapter 4 and the *dkitrc(1M)* manual page for more details.

If you use a **dkuidtab** other than the default, it must be created by **root** with mode *400*. For example, you can have multiple servers (**dkserver**) with each having its own **dkuidtab**. In order to effectively tailor these files to your configuration and environment, refer to the Manual Pages and to the detailed descriptions of these tables in Chapter 3.

Where To Go From Here

The installation of the CommKit Host Interface is now complete.

If you are setting up RFS, uucp e protocol, or other TLI applications, refer to the section *Using TLI Support* in Chapter 4. If you are setting up **uucp d** or **g protocol**, refer to the section *Configuring uucp with d or g Protocol* in Chapter 4. If you are setting up remote printers, refer to the section *Printer Administration* in Chapter 4.

Note: RFS is not supported by the NCR UNIX System.

Remove the CommKit TLI Package

If you are removing the CommKit Host Interface software, you must first remove the CommKit TLI package. To remove the TLI package:

1. Log on as **root**.
2. Execute the following command to verify what listeners, if any, are active.

```
# nlsadmin -x
```

3. Stop all active listener processes by using the **sacadm** command. For example, for a listener started with a **pmtag** (netspec) of **dktp0**:

```
# sacadm -k -p dktp0
```


4. Enter the following command to remove the TLI package:

```
# pkgrm cktli
```

5. If the package displayed is correct, enter **y** to confirm removal.
6. Once the TLI package has been removed, the system will remake a new UNIX Operating System.
7. We recommend you reboot the system, but it is not necessary. However, after removing *all* packages, the system should be rebooted by entering:

```
# shutdown -y -i6 -g0
```

Remove the CommKit Host Interface Software

If you have installed the TLI Support package, you must remove it before you can remove the CommKit Host Interface Software.

Non-Prompted Mode (_dkhost_parms File)

Note: Refer to the section *Installation and Removal Using the _dkhost_parms File* for complete details on the non-prompted mode.

If you create a file */etc/_dkhost_parms* to specify the value of the installation / removal parameters, the removal program will not prompt for them. Refer to Table 2-1 for the shell variables used by the removal procedure.

(Refer to Figure 2-1.) The removal procedure will be terminated if the format or content of the file is invalid. If the */etc/_dkhost_parms* file exists, the following message will appear:

```
The removal parameter file /etc/_dkhost_parms exists.
This removal will be run in NON-PROMPTED mode using the
values specified in this file.

Hit <DEL> key within the next 30 seconds if you do not wish to
remove in NON-PROMPTED mode.
```

If you hit the **Delete** key, the following message will appear:

```
Removal will be run in PROMPTED mode.

Do you wish to continue the removal?
Enter y or n:
```

Enter **n** if you wish to examine the *_dkhost_parms* file.

To remove the CommKit Software using the non-prompted mode, follow the appropriate procedures described under *Prompted Mode* below.

Prompted Mode

To remove the CommKit Software Release 4.4:

1. Log on as **root** on the console.
2. Stop the server(s) by entering the following command for each running **dkserver**:

```
# /opt/dk/sbin/dkserver -t -i interface -s servername
```

interface is the interface board number.

servername is the name of the dkserver (default is the system name [*uname(1)*]).

3. Notify all users that the CommKit Host Interface is coming down by entering the following:

```
# wall
The CommKit Host Interface is coming down.
Please log off if you are using it.
Ctrl-d
```

4. Terminate the **dkdaemon** process or processes where *pid_of_dkdaemon* is the process id of the daemon process.

```
# kill -9 pid_of_dkdaemon
```

Note: Be sure to kill all of the **dkdaemon** processes. There may be several since the administrator can start separate **daemon** processes for each CommKit Host Interface board and driver subsystem.

5. To remove the CommKit Software proceed as shown in Figure 2-9.

Note: Figure 2-9 shows the screen displayed if you saved the customized tables (**dkhosts, dkdotab, dkgroups**) and the service files under */etc/opt/dk/srvtab* directory during the removal of the previous release of CommKit Software.

Figure 2-9: CommKit Software Removal – srvtab as a Directory

```
# pkgrm commkit

The following package is currently installed:
   commkit          CommKit Host Interface to AT&T Data Switch
                   (u3b2v4) 4.4 v3

Do you want to remove this package [y,n,?,q] y

## Removing installed package instance <commkit>

Copyright (c) 1995 AT&T
All Rights Reserved
## Verify package dependencies.
## Processing package information.
## Executing preremove script.

Do you want to remove files /etc/opt/dk/dkhosts,
/etc/opt/dk/dkdotab, /etc/opt/dk/dkuidtab, /etc/opt/dk/dkgroups, and
all files under /etc/opt/dk/srvtab directory (y or n)? n

Do you want to remove the file /etc/init.d/dkitrc (y or n)? n

/etc/init.d/dkitrc will be retained.
```

6. If you enter **y** in response to `Do you want to remove files...`, all of your customized files will be lost. Enter **n** if you wish to keep these files for future use. The system will respond as shown in Figure 2-10.

7. The system will ask if you want to remove the */etc/init.d/dkitrc* file (Figure 2-9). Enter **n** if you wish to save it.

Figure 2-10: Typical Customizing Files Screen

```
Removing files under /var/opt/dk/log directory
Removing the device files
Removing dkpe module ...
Removing dkhs module ...
          •
          •
          •
## Removing pathnames in <tab> class
## Removing pathnames in <none> class
/var/opt/dk/include/sysexits.h
/var/opt/dk/include/stats.h
/var/opt/dk/include/remfio.h
          •
          •
          •
/opt <non-empty directory not removed>
/etc/rc2.d <shared pathname not removed>
/etc/rc1.d <shared pathname not removed>
/etc/rc0.d <shared pathname not removed>
/etc/opt/dk <non-empty directory not removed>
          •
          •
          •
/dgn/X.DKPE
/dgn/DKPE
/dgn <shared pathname not removed>
## Executing postremove script.

Restoring /usr/lib/libdk.so
Updating /etc/ttysrch

## Updating system information.

Removal of <commkit> was successful.

*** IMPORTANT NOTICE ***
        If removal of all desired packages is complete,
        the machine should be rebooted in order to
        ensure sane operation. Execute the shutdown
        command with the appropriate options and wait for
        the "Console Login:" prompt.
```

8. Execute the following to reboot the system:

```
# cd /
# /etc/shutdown -y -i6 -g0
```

3 Control Tables

Introduction

This chapter describes the control tables that must be edited to reflect your AT&T data switch network environment. These tables are used when a call is received or sent through the CommKit Host Interface. The CommKit Host Interface software relies on these tables to implement call validation for such services as login, remote login, remote execution, and file transfer. The control tables also specify the information required to process these services. The CommKit Host Interface tables (residing in */etc/opt/dk*) are:

dkdotab	used for outgoing calls	**dkuidtab**	used for incoming calls
dkhosts	used for outgoing calls	**srvtab**	used for incoming calls
dkgroups	used for outgoing calls		

Overview of Control Tables

When using the **dk** command, the **dkhosts** table is scanned for a match of the host name and service type (**l** for remote login, **x** for remote execution) with the *host* and *classes* fields in the **dkhosts** table. If a match is found, the *Dialstring* and *Miscellany* fields are used to construct the dialstring that is sent to place the call. In addition, if the shell variable DKGROUP is set, the *dkgroups* file is scanned to determine the interfaces and channel numbers to use when establishing the call.

When an incoming call is received, the **srvtab** table is scanned by the **dkserver** program until a match is found on the *system*, *service*, and *user* fields with the same fields of the incoming *dialstring*. When a match is found, the remaining fields in **srvtab** determine what program is to be executed on that channel (such as ***login, uucp***, **push, pull**, etc.). If a match is not found, the incoming call is denied access to the host.

The **dkdotab** table is parsed when the **dkdo** service is invoked on the local host, looking for a match on the *command* field in the **dkdotab** table with the *command* argument to **dkdo**. When a match is found, the remaining fields indicate on which host to execute the *command*, what flags affect the operation of **dkdo**, and what files are associated with the *command*.

Finally, the **dkuidtab** table is scanned on incoming calls to validate and map incoming user IDs with valid user IDs on the local system. The **dkuidtab** table is scanned if the value of the *user* field in the **srvtab** table is an **&** or if the incoming call request includes a DKKEY. A three-field entry is added to the **dkuidtab** table when a user authorizes to the host [see *dkauth*(1C) and *authorize(1M)*]. A *dkuidtab* file is created under */etc/opt/dk* during the installation procedure. The software

distribution medium does not contain a sample of this file. Manual administration of the *dkuidtab* file is unnecessary.

Data Switch Dialstrings

Understanding the purpose and format of the data switch dialstring is crucial to setting up the interface tables. This section gives an in-depth description of the dialstring and its use.

The dialstring is used to access other data switch devices (such as terminals, printers, and other hosts) from your host system. Dialstrings are originated by a sender (such as a terminal user or a program on the local host), expanded by the data switch node, and interpreted by the receiver (such as a server on the remote host). *Dialstring* refers to the sequence generated by the sender and the sequence generated by the data switch node and made available to the receiver.

The format of the dialstring made available to the receiving end differs depending upon whether the node is a BNS, Datakit II VCS, or Datakit VCS node, for different generics of the data switch, and on whether the destination is a host connected by means of the CommKit Host Interface. The following description assumes a Datakit II VCS or BNS data switch.

The format of the expanded dialstring transmitted from the data switch node to the host is as follows:

```
Chan.Token.Lflag.URPinit.Recbuf\n
User-dialstring\n
User-Id\n
Origin.Node.Mod.Ochan.Cflag[.Par1.Par2...]\n
Module Type Information \0
```

Note: The Datakit VCS dialstring is the same, except that it does not include the *URPinit, Recbuf,* and *Module Type Information* fields.

The fields are:

- *Chan* – the channel number assigned to the incoming data switch call (such as a call from the local to the remote host) by the data switch node.
- *Token* – a unique request identifier associated with the incoming call.
- *Lflag* – the local flag byte (**L** if the call originated on the local data switch node and **R** from a remote data switch node).
- *URPinit* – the URP initialization byte (**0** if the call originator handles URP initialization, **1** if the local host is to send INIT1, **2** if the local host is to send INITREQ, and **3** if the local host is to send both INITREQ and INIT1).
- *Recbuf* – the logarithmic (base 2) value of the call originator's receive buffer size.
- *User-dialstring* – the information provided by the caller on the local host in the original dialstring (such as *location, service, protocol,* and *parameter* information).
- *User-ID* – a sequence of characters which the data switch node extracts from the call setup message. The *User-ID* convention is as follows. If the first character is **0**, it represents an octal user ID. If it is entirely numeric, it represents a decimal user ID. Otherwise, it is interpreted as a character login.
- *Origin* – the data switch node group name of the originator as known to the remote data switch node.
- *Node* – the name of the remote data switch node.
- *Mod* – the module number of the call originator in character decimal format.
- *Ochan* – the backplane channel number of the call originator on the module indicated by *Mod*.
- *Cflag* – the call flag (**F** if this is the first call from a port with a predefined destination and **P** if this is the second or succeeding call to this host from the same host with no drop in the incoming *DTR/CARRIER* lead).

- *[.Par1.Par2...]* – reserved for nonpositional parameters. The current implementation supports only a baud rate field which identifies the baud rate of the incoming call (BD=xxxxx).
- *Module Type Information* – the hardware module type and service type of the call originator. It is intended to be used internally by the network.

This discussion will address the *User-dialstring* portion of the dialstring.

Note: The *Chan*, *Token*, *Lflag*, *URPinit*, *Origin*, *Node*, *Mod*, *Ochan*, and *Cflag* values are supplied by the data switch node and are not discussed here.

The format of the *User-dialstring* portion of the dialstring is:

```
location.service.protocol.parameter
```


Although the specific dialstrings can vary, the format of the *User-dialstring* follows these general rules:

- The *location* is of the form *area/exchange/host* where *area* is the name of the area code in which the data switch node resides, *exchange* is the name of the exchange in which the data switch node resides, and *host* is the name of the local service name for the host as defined in the data switch control computer database. This location information is used to route the call through the data switch network to the desired *host*.
- The next fields in the *User-dialstring* are optional to the data switch. Translation of these field values is positional and dependent on the program processing the incoming call message, such as the CommKit Host Interface server **dkserver**. The incoming dialstring is translated on the remote host using the **srvtab** table.
- The *service* field contains the actual CommKit Host Interface service desired. This field is mapped into the *service* field in the **srvtab** table on the remote host.
- The *protocol* field contains optional protocol specifications. The values for the *protocol* field of the dialstring map into the *flags* field in the **srvtab**. These protocol values can be grouped logically as follows:

- Protocol values indicating the type of service (that is, *raw*, *terminal*, and *remote execution* services, etc.) are to be used on the remote host
- Protocol values applicable to use on the remote execution driver
- Protocol values applicable to use for the terminal driver
- Other flags.

These protocol field specifications are optional and, if not specified by the user, can be supplied by the calling program on the local host or mapped from the entry in the **srvtab** on the receiving host.

- The *parameter* field contains an optional parameter field interpreted by the program that processes incoming service requests such as **dkserver**.

Examples

If a user with a login of **goldfish** having a numerical user ID of **104** is logged on to the host **fish** and wants to remotely log on to host **bird**, the following command would be executed on **fish**:

Note: All command entries must be followed by the Enter key.

```
dk nj/shore/bird
```

The **dk** program calls **maphost** to add the *service* and *protocol* field specifications to the outgoing dialstring. The dialstring now becomes:

```
nj/shore/bird.rl.vx
```

- *nj/shore/bird* is the location or *area/exchange/host* information

- *rl* is the service name for remote login [see *srvtab(4)*]
- *v* and *x* are protocol values translated on the receiving (remote) end as:
  - *v* Environment variables should be read from the incoming data channel
  - *x* Open the remote execution protocol device driver for this channel.

The data switch node then adds routing information to the supplied dialstring, and the receiving host sees the following dialstring arrive for service:

```
Chan.Token.Lflag.URPinit.Recbuf\n
nj/shore/bird.rl.vx\n
104\n
Origin.Node.Mod.Ochan.Cflag[.Par1.Par2...]\n
Module Type Information\0
```


- *Chan* is the channel number assigned by the data switch node and subsequently translated into a UNIX system device on the remote host
- *nj/shore/bird* is the *location* provided by the caller on the local host in the original dialstring
- *rl* is the service to be executed on the remote host (as supplied by the **dk** program)
- *v* and *x* are the protocol values (as supplied by the **dk** program)
- *104* is the numerical user ID of the user **goldfish** on the calling **fish**
- *Origin* is the originating group
- *Token*, *Lflag*, *Node*, *Mod*, *Ochan*, *Cflag*, *URPinit*, and *Recbuf* are the keywords associated with the values supplied by the data switch node.

The **dkserver** running on the remote host will process incoming call requests. When the call comes in on the serving channel from the data switch node, the server uses the incoming dialstring to interrogate the server table to determine what action should be taken to satisfy this service request. It uses the *system* (**nj/shore/bird**), *service* (***rl***), and *user* (**104**) fields to find a match with an entry in **srvtab** so that the correct action can be taken. The server table entry shown below matches the incoming call.

```
  System       Service   Flags   User   Program    Initial Parms
-------------------------------------------------------------
nj/shore/*    rl      /vaex    *n      %s      -Xsh:-c:%p
```

If *protocol* values of the *flags* field had not been supplied (which is not the case in this example), then the *protocol* values **v**, **a**, **e**, and **x** would have been supplied. The program executed on the remote host is **%s**, which is translated by **dkserver** into the pathname of the user's shell as specified in the */etc/passwd* file. ***n** requests that the system use the incoming numeric user ID (**104**) transparently if an entry is found in */etc/passwd* with the same number.

The parameters to the shell are **-Xsh:-c:%p**. **%p** is translated by the server as any parameter sent in the dialstring from the local host, reparsed, so that if it contains colons, separate arguments will be generated. The server will then ***fork*** and ***exec*** a copy of the program supplied in the *program* field with the parameters supplied in the *Initial Parms* field to handle this call. If the remote host's password file does not contain a valid entry for user **104**, the call is rejected with an *access denied* error code.

dkhosts

Refer also to the *dkhosts(4)* manual page.

The CommKit Host Interface software package includes a sample **dkhosts** table found in the directory */etc/opt/dk*. Edit this sample table to reflect your network environment.

Additionally, the user may create a *dkhosts* file in his/her home directory (*$HOME/.dkhosts*).

The **dkhosts** mapping table is used for expanding dialstrings when making outgoing calls from the host. The table is presented as shown:

```
host   classes  dialstring  miscellany
```

- *host* is the name given to a remote host
- *classes* define the services (remote login, remote execution, file transfer) supported by the remote host
- *dialstring* is the full data switch address of the remote host
- *miscellany* includes the arguments for the remote service.

Note: Delimit fields with tabs only; do *not* use blanks.

When creating a dialstring, the **dkhosts** interface table is sequentially scanned for a match in the *host* and *classes* fields. If a match is found, then the *dialstring* and *miscellany* fields are used to create the dialstring.

The following service *classes* are supported on the host:

d	**dkdo** service	l	Remote logins	x	Remote execution
f	File transfer	p	Printer		

For example, if the host **fish** offered **d**, **l**, and **x** [see the *dkdo(1C)* and *dk(1C)* manual pages], then the entry for **fish** would be:

```
fish        dlx       nj/shore/fish         -
```

The *miscellany* field is present for cases where the call through the CommKit Host Interface is nonstandard or for cases that require additional passed parameters. The *miscellany* field entries contain subfields separated by commas. These subfields contain two-character names constructed from the service *class* character and another character appropriate to the subfield value. The characters representing the field names are:

c	alternate command to execute locally
s	service name
v	existence of environment variables (**y** or **n**)

p protocol string
o old protocol (**y** or **n**)

An example use of the *miscellany* field is to force the **dk** command to use **dkcu** for remote logins to a particular host. This would be necessary if the destination host does not have a remote execution driver and, therefore, does not support the remote login protocol. In this case, the *miscellany* field for the desired *host* is parsed, and the retrieved information is used to complete the call.

```
lc=dkcu:-f,lp=t
```

An example of an entry in the *dkhosts* file is given below:

```
fish  dlx   nj/shore/fish    lc=dkcu:-f,lp=t
```


This entry is for host, **fish**, which can handle calls defined as follows:

Host	Service Class	Dialstring	Command Specification for Remote Login
fish	**d, l, x**	**nj/shore/fish**	**dkcu -f** TTY (t) protocol on a remote host

Briefly stated, for an **l** service *class* entry for **fish**, the **dkcu** command (**c**, alternate command to execute locally) should be executed with an argument of **-f** and uses **t** (**p**, protocol string) protocol.

Another example entry in the *dkhosts* is:

```
bird   flx   nj/shore/bird    -
```

This entry is for host, **bird**, which can handle calls defined as follows:

Host	Service Class	Dialstring
bird	**f, l, x**	**nj/shore/bird**

A call being made to **bird** with any one of the service *class* values in the table above will use the dialstring shown. **bird** is on data switch node **shore**, and the local data switch node is in **nj**. The *miscellany* field entry is a "-" which indicates that there is no special processing that must be done to put this call through the CommKit Host Interface.

Note: This dash must be included as a place holder or the entire entry will be ignored.

Edit the sample *dkhosts* file to reflect your own data switch network environment. Change the host names to reflect the proper host names that will be referenced in a CommKit Host Interface call, and the full dialstrings mapped accordingly. For each host attached to the data switch node, determine the service *classes*.

In general, the hosts that are local to your environment will support all the above services. The remote hosts that are accessed through dialers will probably support only remote logins, and the call might be made by means of **dkcu**. The entries in the *dkhosts* file for access to remote hosts often depend on the hardware connections to those hosts and on the path taken to access them.

The *host* field corresponds to the host name being referenced. (In the previous example, a valid host name is **bird**.) The service *classes* field matches the service specified in the incoming *host* and service *classes* combination.

Destination Mapping

The two *dkhosts* files – *$HOME/.dkhosts* and */etc/opt/dk/dkhosts* – are treated as one concatenated file when **maphost** is invoked for destination mapping. **maphost** will first check the entries in the local *$HOME/.dkhosts* file, and then, if no match is found or if the file does not exist, **maphost** will check the */etc/opt/dk/dkhosts* file.

The following shows an example of calling *maphost*(3X) directly from within a C program to perform destination mapping. Its purpose is to show how *maphost*(3X) references both the local and system-wide *dkhosts* files.

The *$HOME/.dkhosts* contains the following:

```
# Host          Classes  Dialstring              Miscellany
hudson          dlf      nj/local/hudson         -
lineprt         p        nj/local/lineprt        -
```

/etc/opt/dk/dkhosts contains the following:

```
# Host          Classes  Dialstring              Miscellany
fraser          dlf      ca/remote/fraser        -
hudson          dlf      ca/remote/hudson        -
edsel           dlf      ca/remote/edsel         -
tercel          dlf      ca/remote/tercel        -
```

The example C program contains the following entries:

```
main()
{
        (void) printf("1 dialstring: %s\n",maphost("",'f',"","",""));
        (void) printf("2 dialstring: %s\n",maphost("edsel",'f',"",""));
        (void) printf("3 dialstring: %s\n",maphost("",'f',"","",""));
        (void) printf("4 dialstring: %s\n",maphost("",'f',"","",""));
        (void) printf("5 dialstring: %s\n",maphost("bingo",'f',"","",""));
        (void) printf("6 dialstring: %s\n",maphost("",'f',"","",""));
        (void) printf("7 dialstring: %s\n",maphost("oh/mid/viper",'f',"","",""));
        (void) printf("8 dialstring: %s\n",maphost("",'x',"","",""));
}
```

The output of the example C program with the example *$HOME/.dkhosts* and */etc/opt/dk/dkhosts* files is:

```
1 dialstring: nj/local/hudson
2 dialstring: ca/remote/edsel
3 dialstring: nj/local/hudson
4 dialstring: ca/remote/fraser
5 dialstring: bingo
6 dialstring: nj/local/hudson
7 dialstring: oh/mid/viper
8 dialstring:
```

The following is a description of the output:

- Line 1: The sample C program contains a call to **maphost** with a null host argument. The local *dkhosts* file, *$HOME/.dkhosts,* is referenced first because it exists, and the first service class of **f** is matched. The local *dkhosts* file remains open.
- Line 2: Because **maphost** was previously called with a null host, successive calls to **maphost** begin with last entry matched. Host name **edsel** is not found in the local *dkhosts* file (*$HOME/.dkhosts*), therefore, the system-wide *dkhosts* file (*/etc/opt/dk/dkhosts*) is referenced. A match is found, and both files are closed because a host was specified.
- Line 3: **maphost** is called with a null host. The local *dkhosts* file is opened, and a match is made on the first service class of **f**. The file remains open for successive **maphost** calls since a null host was specified and a match was found.
- Line 4: The next call to **maphost** with a null host fails to match on a service class with the remaining entries in the local *dkhosts* file. The system-wide *dkhosts* table is opened, and the first entry matches the service class passed by **maphost**. Both files remain open.
- Line 5: Because both tables are open, **maphost** will begins its search starting with the last entry matched on in the system-wide *dkhosts* file. Because no match is found, the host name is returned and both tables are closed.
- Line 6: **maphost** is called with a null host. The local *dkhosts* file is opened and returns the first service class entry matched. The file is kept open because a null host was specified and a match was found.

- Line 7: **maphost** is called with a host name which contains slashes and is immediately returned. No searches are performed in either table, but the local *dkhosts* table is closed.
- Line 8: **maphost** is called once again with a null host. The local *dkhosts* file is opened. No match is found with a service class of **x**. The system-wide *dkhosts* file is then searched. Because no match is found, both files are closed and NULL is returned.

dkgroups

Refer also to the *dkgroups*(4) manual page.

The CommKit Host Interface software package includes a sample **dkgroups** table found in the directory */etc/opt/dk*. Edit this sample table to reflect your network environment and / or channel limit imposed on user groups through the use of the shell variable, DKGROUP.

The system administrator can use the shell variable DKGROUP to control the available interfaces and channels on a per-user basis by setting DKGROUP in */etc/profile* and making it a read-only shell variable.

The **dkgroups** channel group table is used to provide limits on the number of channels a group can use. This feature can be used in conjunction with multiple groups on a CPM-HS to provide multiple originating groups. When used without multiple groups on a CPM-HS, **dkgroups** provides a method to limit the number of channels a group can use on a CommKit Host Interface; when used with multiple groups, it provides a method of controlling the originating group used in the **srvtab**(4) to control access permissions.

For more information on multiple groups associated with a CPM-HS module, refer to the Commands Reference guide for the data switch you are using (Datakit, BNS-2000, etc.).

The **dkgroups** table format is as follows:

```
name    interface       low     high
```

- *name* is the name associated with the **dkserver** channel range. This name does not have to match the group name on the data switch.
- *interface* is a comma-separated list of interfaces on which the channel group is defined.
- *low* is the lowest channel for the channel group. When using multiple interfaces the channel range is the same across all interfaces.
- *high* is the highest number channel in the channel group. When using multiple interfaces the channel range is the same across all interfaces.

The *low* and *high* values can be the same, which defines a channel group of only one channel. The channels are reserved from the lowest to highest channel, inclusive.

The following example defines a channel group named **onechan** with a single reserved channel (**53**) on interface **1**.

```
onechan 1       53      53
```

When DKGROUP=onechan, all calls will use the special file */dev/dk/1.053* to make calls. While a call is actively using this group, all other call attempts will fail with the error EX_UNAVAILABLE.

The example below defines a channel group named **multintf** with 10 reserved channels on two interfaces.

```
multintf        1,3     10      19
```

This could represent a situation in which the two interfaces are backups for a fault-tolerant system. When DKGROUP=multintf, call attempts will first try interface **1** using channels **10** through **19**, and then try interface **3** using channels **10** through **19**.

After editing */etc/opt/dk/dkgroup*, the channels must be reserved by means of *dkmaint*(1M). This can be done manually, by executing *dkmaint* for each group defined in *dkgroups,* or by stopping and then starting the CommKit Host interface servers by means of *dkitrc*(1M). *dkitrc* will automatically reserve the channel groups.

Caution: **Take care in adding channel groups in *dkgroups*. Channel 0 or 1 should *never* be a channel group or part of a channel group.**

srvtab

Refer also to the *srvtab(4)* manual page.

The granting and denial of access privileges by the CommKit Host Interface on a called host is controlled entirely by the *server table* on that called host. An incorrect or incomplete server table can cause serious security problems by allowing unauthorized access to system files and resources.

This section describes the facilities available through the **/etc/opt/dk/srvtab** server table and provides the user with several suggestions to make the network connections more secure. A high degree of security is ensured by editing server table files.

Caution: **The sample */etc/opt/dk/srvtab* files distributed with the CommKit Host Interface software are not intended to be used as provided and do not provide the customer with a high degree of security as the default. The customer must customize the sample server table**

files to achieve the desired level of security.

Server Table

The directory */etc/opt/dk/srvtab* and associated files (referred to as the server table) are used to validate incoming call requests and map them into processes on the called host. The server table can be tailored to restrict the types of calls permitted. All incoming calls must be mapped by means of the *server table*; there are no privileged calls that can bypass this procedure.

The server table is a directory containing files whose names correspond to the names of requested services. For example, */etc/opt/dk/srvtab/pupu* is the name of the file used for the file transfer service, **pupu**. Alternatively, the server table may be configured by the user as a single ASCII text file consisting of comment lines and mapping entries. Entries may not span lines. In structure, this server table format resembles a concatenation of the files normally found in the */etc/opt/dk/srvtab* directory.

Comments are indicated by a "#" character in the first column and are ignored during call validation and mapping. Use comments to describe the function of mapping lines. You may also use comments to disable mapping lines without removing them from the file.

Mapping lines in the **srvtab** table consist of six tab-separated fields:

```
system     service     flags     user     program     initial parms
```

Note: Delimit fields by tabs only. Do not use blanks.

These fields are described below.

System Field

The *system field* (originating group) contains a pattern against which originating groups are matched. The format of the pattern is **area/exchange/group[!][.user]**, where the suffixes **!** and **.user** are optional.

The **area/exchange/group** pattern may be specified in one of the following ways:

- completely specified as in **lc/sporty/hotrod**
- consisting of a single asterisk (wild card) as in *****
- consisting of a partially specified originating group with embedded wild card characters:
  - — asterisk which matches any set of zero or more characters
  - — question mark which matches any single character
  - — brackets, which specify specific listed characters, a range of characters that match a single character (by means of a dash **-**), and / or a negated list indicated by the symbol **^**.

Under these rules:

- Originating group fields **l*?/sp?rty/h*rod***, **??/*/hotrod**, and **lc/spor???h*** all match the originating group **lc/sporty/hotrod**
- Originating group fields **l??/sp?rty/h*rod***, **??/*x/hotrod**, and **lc/spor??h*** do *not* match the originating group **lc/sporty/hotrod**

  Note: The originating group fields ***,*/***, and ***/*/*** are equivalent and match *any* originating group.

- The originating group field **l*?/sp[ao]rty/hotrod[b-df]** does *not* match the originating groups **lc/sporty/hotrod, lc/sporty/hotrod2,** but *does* match **lc/sparty/hotrodb, lc/sparty/hotrodc, lc/sparty/hotrodd,lc/sparty/hotrodf lc/sporty/hotrodb, lc/sporty/hotrodc, lc/sporty/hotrodd, lc/sporty/hotrodf**.
- The originating group field **lc/sp[^e]rty/*[^1a-c]** matches **lc/sparty/hotrodd, lc/sparty/hotrodf,** and **lc/sparty/hotrod2**, but not **lc/sperty/hotrod1, lc/sperty/hotroda, lc/sperty/hotrodb,** or **lc/sperty/hotrodc**.

The originating group suffix **!** may be specified for originating group patterns that end with an asterisk (*****) wild card. The exclamation point (!) prevents the asterisk wild card from matching the local *dkserver* name. For example, if the

local server name is **camaro**, the pattern **lc/sporty/*!** will match all originating groups in *area/exchange*, **lc/sporty** except **lc/sporty/camaro**. This feature prevents users from calling themselves on the same host. The local server name and the group name of the host in the data switch control computer database must be the same for the **!** feature to work.

.user Suffix

The originating group suffix **.user** may be used to match specific users from either specific groups or from classes of groups. The **user** must be specified as a decimal integer user ID to match the format provided in the call request information. The group pattern ***.0** would match the user **root** from any originating group, while the pattern **lc/sporty/*.0** would match the user **root** from any group in the *area/exchange*, **lc/sporty**. A question mark is used to match the null user ID. Since **TY6** and **TY12** ports do not generate a user ID as part of the dialstring, the pattern ***.?** may be used to match requests from an originating terminal or modem group. The pattern **lc/sporty/*!.0** on the host **hotrod** would match the user **root** from any group except **lc/sporty/hotrod**.

The symbols <, >, and **-** may be used as described below.

Note: Invalid user ID formats will be reported in *dksrvlog*.

- The pattern **lc/sporty/hotrod.<uid1** would match any user with an incoming uid *less than* **uid1**. Similarly the pattern **lc/sporty/hotrod.>uid1** would match any user with an incoming uid *greater than* **uid1**.
- The hyphen (**-**) is used to specify a range. For example, the pattern **lc/sporty/hotrod.uid1-uid3** will match any user with an incoming uid between **uid1** and **uid3**, inclusive.

  Note: When specifying a range, the first *uid* must be less than or equal to the second *uid* in the range.

Service Field

The *service field* contains the name of the service the caller may request. This field consists of a single word or abbreviation and is used to match the service string that appears after the first period in the requester's dialstring. A single asterisk may be used to match any requested service and the special entry "-" is used to

match the case where no specific service was requested by the dialstring.

You may add additional unique services to the server table, but the following alphabetically listed services are distributed as part of the standard product:

–

The **null** service name is used to match requests not specifying a specific service. This default service normally maps to an invocation of the ***login*** program which is used to provide login initiation services. A user-friendly system in a trusted environment might invoke a directory assistance or application program directly as the default.

*****

The **wild card** service name is used to match any requested service name. This facility may be used in conjunction with the **T** flag (described later) to reject calls by classes of users or from classes of sites.

authorize

The **authorize** service is used to establish preauthorized user name mapping between called hosts. When a local user requests authorization on a remote host, the remote host is requested to invoke the **authorize** service via the **dkauth** command. With **dkauth**, the user can:

- Perform authorizations over multiple interfaces to a specified remote system. The user can verify remote authorization by:
  - — using **dk** or **dkcu** to connect to the remote system and then running **dkauth -d**, or
  - — remotely executing the command:

    dk *system* **dkauth -d**
- Rejuvenate expired authorizations on the local system without having to re-authorize from the remote system.
- Delete authorizations on the local system without having to use the original remote system to delete the authorize entry.

dkauth also allows the super user **root** to see all local authorizations on the system.

See *dkauth(1C)* for the proper way to invoke the **dkauth** command. The pre-authorized user mapping facility is described in more detail in the section *User Field*, later in this chapter.

dkload

The **dkload** service is used by the diagnostic tool **dkload** for interface loading. This service supports the remote end of a loopback test program that can be used to saturate the CommKit Host Interface.

do

The **do** service is used by the command **dkdo** to provide transparent remote execution of commands across different hosts.

login

The ***login*** service maps to an invocation of the ***login*** program which is used to provide login initiation services.

pupu

The **pupu** service maps to an invocation of the **pupu** program which is used to provide the remote support for a **push** or **pull** request.

rl

The ***rl*** service is used to match requests for remote login from another host. The remote login service is normally requested over a remote execution channel and provides the requester with a remote execution of an interactive command interpreter or shell.

Note: This facility allows a requester to bypass the normal login / password sequence when remotely logging on to a host. It is also possible to map remote login requests to an invocation of the ***login*** program and thereby require a requester to supply a valid login and password.

rx

The ***rx*** service is used to match requests for remote execution from another host. The remote execution service should always be requested over a remote execution channel and should specify the desired program to be executed. A remote execution request is very similar to a remote login request except that a program is specified with ***rx*** requests and the program is implicit (an interactive shell) with ***rl*** requests.

uucp

The ***uucp*** service is provided to match requests for ***uucp*** service. This service should map to an appropriate ***uucp*** login (such as **nuucp**) and invokes the ***uucico*** program.

Note: Different login names can be specified, based on the originating group name. For example, **luucp** might be used for requests from local hosts, while **nuucp** would be used for requests from unknown or "network" hosts. See the section *Fixed User ID Mapping*, later in this chapter.

whoami

The ***whoami*** service maps to an invocation of the ***echo*** command and is used to echo back a string showing the requester's user number and originating group name.

Other services in the */etc/opt/dk/srvtab* directory are provided as examples.

Flags Field

The *flags field* of a server table mapping line contains one or more flags that may be used to control or modify the actions the server takes when processing a call request that has been mapped to that line. Flags specify options that are unique to the mapped line. A brief description of each flag is provided in Table 3-1.

Table 3-1: Server Table Flags

Flag	Description
a	Additional arguments should be read from the incoming data channel before execution of the program. This flag should be specified only for remote execution channels (see **x** option).
e	Arranges for the exit code of the program to be passed back to the originating system. This flag should be specified only for remote execution channels.
h	Invokes the mapped program with the SIGHUP hangup signal ignored. See *signal(2).*
l	Spawns a TLI application from the **srvtab** service.
t	Opens the **tty mode** data service and invokes the program with the *stdin, stdout,* and *stderr* files attached to the **/dev/dkt/?.???** device for the tty channel. Note that the **t** flag should be used with caution when it is associated with non-standard service. The *t* flag uses a TTY interface that echoes its input by default. Origination endpoints other than a user terminal (from a DESTINATION prompt or a **dkcu**) will not receive the echoed data. If the data is not drained by the originating endpoint the network will flow control the echoed data, which will build up on the destination host. The UNIX system STREAMS module **ldterm** that actually echoes the data, does not honor flow control and will consume all of the STREAMS resources.
u	Opens the ***raw*** [URP protocol] data service and invokes the program with the *stdin, stdout,* and *stderr* files attached to the **/dev/dk/?.???** device for the raw channel.
v	Environment variables should be read from the incoming data channel before execution of the program. This flag should be specified only for remote execution channels.
x	Opens the remote execution protocol data service and invokes the mapped program with the *stdin, stdout,* and *stderr* files attached to the **/dev/dkx/?.???** device for the remote execution channel.
F	This flag allows the invoked program to be orphaned by *dkserver*(1M). The **F** flag is used to inform *dkserver*(1M) that the invoked program should be a child process of *init*(1M). Use of this flag prohibits the use of any *utmp* related flag: **I, L, U**. If any *utmp* flag is used in conjunction with the **F** flag, the incoming call will be rejected.
I	Creates an INIT_PROCESS type *utmp* entry for the invoked program. This style of accounting makes an entry in the system *utmpx* file that is transparent when using the default options of the ***who*** command. It also makes an entry in the system *wtmpx* file that may be displayed using the command ***last(1C)***. Use of this flag prohibits the use of the **F** flag. If the **I** flag is used in conjunction with the **F** flag, the incoming call will be rejected.

Table 3-1: continued on next page

Table 3-1: *Continued*

L	Creates a LOGIN_PROCESS type *utmp* entry for the invoked program. This style of accounting makes an initial entry in the system *utmpx* file that is transparent when using the default options of the ***who*** command. This style of accounting assumes that the invoked program (e.g., ***login***) will overwrite this entry in the system *utmpx* file with valid user information. This style of accounting does not make an entry in the system *wtmpx* file; it assumes that the invoked program will make the initial entry, thus avoiding multiple entries. Use of this flag prohibits the use of the **F** flag. If the **L** flag is used in conjunction with the **F** flag, the incoming call will be rejected.
M	Sends mail to the specified user ID when a service is requested illegally. In order for the **M** flag to operate, each service file must be modified by adding a line to the end of the file. Refer to the example given in the *srvtab*(4) manual page.
P	Retains the **dkserver**'s *nice*(2) priority when invoking the application. The application is expected to lower the priority after a short interval by means of *nice*(2). If this is not done, incoming call processing can fail because there are too many high priority processes. The **P** flag should *never* be used in a **srvtab** entry that invokes user login processing or any other long running processes. (See Note, below.)
R	Rejects the call unless the resulting mapped user ID has an "ordinary" shell. An ordinary shell is defined as either a null shell field in the password file (the default **/bin/sh**), or a shell field that references a program that ends in the string "sh" and the basename of that program does not begin with the letter **r** (restricted shells like **/bin/rsh**).
T	Traps the call, rejecting it with a NAK code obtained from the first argument of the program arguments field. The remainder of the program argument field will be processed for "special code" expansion and will then be logged to the server log file.
U	Creates a USER_PROCESS type *utmp* entry for the invoked program. This style of accounting makes an entry in the system *utmpx* file that is displayed when using the default options of the ***who*** command. It also makes an entry in the system *wtmpx* file that may be displayed using the command ***last(1C)***. Use of this flag prohibits the use of the **F** flag. If the **U** flag is used in conjunction with the **F** flag, the incoming call will be rejected.
/	If a **/** flag is present, the flags that follow it act as the default flag settings and the user-supplied protocol field portion of the dialstring (if any) replaces the part after the **/**. This flag is invalid if the user-supplied protocol field contains illegal flags or if it contains an **I, L, R**, **T** or **U** flag. The flags for remote execution channels are often specified as **/vaex** to indicate that the default is to honor arguments, environment variables, and to return an exit code.
|	If a **|** flag is present, the list of STREAMS modules that follows the **|** will be pushed onto the *stdin* STREAM. The STREAMS module list should be separated by colons (:)

Table 3-1: continued on next page

Table 3-1: *Continued*

	if more than one STREAMS module is specified. The | flag should be the last flag specified in the flag field.
E*t*	The **E***t* flag may appear only in the dialstring as the final two characters of the protocol field. **dkserver** sets the endpoint type to *t*, overriding the default value provided by the data switch network. The character *t* must be a valid data switch endpoint type code. See *dkepoint(3X)* for additional information.

Note: When using the **P** flag, the invoked application should return the *nice* value to a neutral value by using a code fragment similar to the following:

```
void restore_priority()
{
     int curr_nice;

     /* get current nice value by adding zero to it */
     curr_nice=nice(0);
     /*make the current value zero. */
     (void)nice(-curr_nice);
}
```


User Field

The *user field* (user ID mapping) of a server table entry determines the way in which the calling user ID is treated. The user ID in the call request may be passed to the program unchanged, translated (mapped) into a new user ID, restricted to a range, or ignored altogether by using a fixed user ID. See Table 3-2.

Regardless of the user ID mapping mode specified, a valid */etc/passwd* entry must exist for the resultant user ID in order for the server table entry to match the call request. A program **cannot** be invoked with an invalid or illegal user ID.

Table 3-2: User ID Mapping Options

Option	Description
***n, *o**	Use the numeric user ID supplied in the call request information. The ***o** means the user ID should be interpreted as an octal number. The ***n** indicates the user ID should be treated as a self-determining number. For example, an initial **0x** or **0X** indicates hexadecimal, and an initial **0** indicates octal. The ***n** and ***o** user ID formats will only provide a valid match if an */etc/passwd* file entry exists with the same *numerical* user ID and the password has not expired. The group id is determined from that password file entry.
&	Translate the supplied user ID and group-id names using the *dkuidtab* file. This type of entry matches only those user IDs which have entries in the *dkuidtab* file. This facility allows a user with a user ID on one host to be preauthorized as a user with a different numerical user ID on another host. This preauthorization is performed by the **authorize** command.
<uid, >uid	The previous two forms can be further restricted to a range of user IDs by appending *<uid* or *>uid* to the field. This restricts the incoming user ID to be less than (or greater than) the specified decimal number. For example, "***n>0**" prevents **root** (user ID 0) from matching the line. Only one modifier may be appended to an entry.
[login]	This type of entry provides a fixed login name that is used for all matching call requests. The [***login***] user ID format will return a valid match as long as a valid */etc/passwd* entry exists for ***login***. Fixed login name specifications are useful for assigning a single ***uucp*** login to a group of originating sites. The facility is also useful when invoking authorization type services that require a fixed set of permissions. The [***login***] login ID format can be thought of as a form of **setuid** facility, since all call requests are mapped to the same login ID.
= and **-**	The symbols **=** and **-** are allowed to specify a specific *uid* or a range of *uids*, respectively. Examples are shown below: *n=uid access granted for a specific uid *n=uid1-uid2 access granted for a range of uids *o=uid access granted for a specific uid *o=uid1-uid2 access granted for a range of uids &=uid access granted for a specific authorized uid &=uid1-uid2 access granted for a range of authorized uids

Program Field

The *program field* of a server table entry contains the pathname of the program to be executed. The field may contain a **%s** which will be replaced by the pathname of the user's shell as obtained from the */etc/passwd* file.

The server uses the *execv* library function when invoking programs, so only paths to binary executables may be specified in the *program field* of a server table entry. The path should be fully specified (for example, */opt/dk/bin/program*) for each entry in the server table.

Initial Parms Field

The *initial parms field* (*fixed program arguments field*) of a server table entry specifies the initial (fixed) arguments for the invoked program. The colon-separated arguments from the server table will be passed to the invoked program starting as the zero argument. If the flag is present in the server table entry, additional arguments obtained from the incoming remote execution channel will follow the initial arguments when the program is invoked.

Initial arguments may consist of, or be embedded with, one or more tokens (sometimes referred to as *special codes*) that may be used to substitute information from the dialstring. Each token is a two-character string consisting of a percent sign (%), followed by a single alphabetic character. Invalid tokens are silently removed and are not expanded.

Table 3-3 summarizes the values and descriptions of the allowed substitution tokens. Since some older versions of the *Datakit* II VCS common controller do *not* support certain features, the substitution string may turn out to be null.

Table 3-3: Program Arguments Specification

Specification	Description
%b	The baud rate of the calling terminal. This token is not expanded if the incoming call did not originate from a port providing baud rate information. (For example, hosts currently do not provide this).

Table 3-3: continued on next page

Table 3-3: *Continued*

Specification	Description
%c	Originating channel number of the call being serviced.
%d	The dialed server name (for example, **host**). This does not include the period or anything after it.
%e	The service field of the dialstring (for example, **pupu**).
%f	The originating group name of the caller (for example, **lc/sporty/hotrod**).
%h	The local server name.
%l	The originator, as known to the local node.
%m	The originating module number of the call being serviced.
%n	The originating data switch node name.
%o	The type of originating device.
%p	Parameters from the dialstring. The parameters are reparsed so that colons may be used to produce separate arguments.
%r	The protocol field of the dialstring, if any.
%s	The pathname of the user's shell as obtained from the */etc/passwd* file. A null shell field in the password file will be expanded as **/bin/sh**.
%t	The device file name that corresponds to the assigned CommKit Host Interface channel, minus the initial /**dev**/.
%u	The numeric user ID of the user placing the call.
%x	The call flag applies only to originating data switch ports (for example, **TY6**, **TY12**, etc.) which have been assigned a *predefined destination.* An **F** is returned in place of the flag if this is the first call from the originating device and a **P** if there have been previous calls.
%z	The module type flag will return the module type of the originating device if the data switch includes this information in the dialstring (field 1 of the fifth line of the dialstring).
%C	The server control file (for example, **srvtab**) or directory name used.
%H	The originating group name truncated to the length of the host field of an */etc/utmp* entry.
%U	The user ID mapping file name (for example, **dkuidtab**) used.

Server Table Scanning Rules

Several parameters are associated with each incoming call. Some of these parameters include the requester's originating data switch group, the requested service name, and the requester's user ID, if known. The server uses these parameters to match lines in the server table during the call validation process.

Modifications to the Server Table

The server table is opened each time a call request is received from the data switch node. Opening the server table for each incoming call allows the administrator to make modifications that will take effect on the next incoming call. There is no need to restart the server after modifications to the table.

Server Table Validation and Matching

Since */etc/opt/dk/srvtab* is a directory, the file corresponding to the requested service is examined. If that file does not exist, the wild card file ***** is used. Call requests are denied with an access denied [see *dkerr(3x)*] rejection code if the end of the table is reached before a match occurs. All lines containing a **#** character in the first column and all lines without the proper number of fields are ignored. Following the format validation of a server line, three comparisons are made with the call request information:

1. The requested service is compared to the service specified in the server table entry. If the services do not match, the scan moves on to the next line of the server table.
2. The originating group for the call is then compared against the originating group pattern in the server table entry according to the rules specified in the *System Field* section. If the originating group fails the check, the scan moves on to the next line.
3. Finally, the originating user ID contained in the call request information is processed by the method specified in the *user field* of the server table entry. If the user ID processing results in an invalid user ID on the called host, the scan moves on to the next line.

The first server table line that passes all the specified tests is considered a match. When a match occurs, the call is accepted and the program specified in the server table entry is invoked with the appropriate arguments according to the server table flags.

Group.user Facility

The **.user** suffix to the originating group pattern is a useful facility for restricting access by certain (often privileged) users. However, you must set up the server table carefully or the desired effect may not be achieved.

An example server table fragment, that illustrates an actual problem one administrator had using the **.user** originating group pattern suffix, is shown in Figure 3-1.

Figure 3-1: Server Table Fragment .user Example

```
lc/sporty/camaro pupu -  *n     /opt/dk/bin/pupu pupu:from:%f
*.0              pupu -  guest  /opt/dk/bin/pupu pupu:from:%f
lc/sporty/*!     pupu -  *n     /opt/dk/bin/pupu pupu:from:%f
*                pupu -  &      /opt/dk/bin/pupu pupu:from:%f
```


If a **pupu** call request comes from **root** in originating group **lc/sporty/camaro**, the request will match the first server table entry and will be run as **root**. The ***n** user ID option indicates that no user ID translation will take place.

If the password file contains a login with the symbolic name **guest** and a **pupu** call request comes in from **root** in an originating group other than **lc/sporty/camaro**, the request will match the second server table entry and will use the login **guest** for the **pupu** transaction.

If the called host has no **guest** login in the password file and a **pupu** call request comes in from **root** in an originating group other than **camaro** in the *area/exchange* **lc/sporty**, the request will match the third line of our example server table and will allow the call to take place as **root**. The second line of the server table is invalid in this case because there is no **guest** in the password file.

Finally (still assuming no **guest** entry in the password file), if a **pupu** call request comes in from **root** in an originating group outside the *area/exchange*, **lc/sporty**, the request will match the last line of the example server table and the call will be denied unless **root** on the requesting host has a valid authorization. See *authorize(1M).*

The example in Figure 3-1 shows that considerable care must be given to the specification of server table entries. The lack of a **guest** login on the system prevented **root** requests from being mapped to **guest** even though the administrator may have been successful in employing this server table fragment on a different system. The administrator may have removed the **guest** login to limit exposure from a different area without realizing the impact on the server table.

User ID Mapping Rules

The *user field* (user ID mapping) of the server table provides a flexible mechanism for restricting or translating call requests from classes of users. Calls may use *transparent*, *translated*, or *fixed* mapping. Transparent and translated mappings may be further restricted to numerical user ID ranges. The type of mapping may be selected individually for each originating group and service combination, or may be specified using originating group patterns.

A user ID mapping facility is necessary because every user on every remote host your system communicates with may not have a login on your system. Some of your systems may have identical password files (for example, each user has the same login on each system), while other groups of systems may have no users in common. Commonly administered systems in a computer center may have administrative logins (for example, **root, bin, sys**) in common, while each user login may be unique to a single system. Finally, numerical user IDs (for example, the user number for each login in the */etc/passwd* file) may be unique across all of your systems (for example, user ID 41627 identifies the login for "Rogers" and is only used on systems where a **rgrs** login exists) or the numbers may not be unique (for example, user ID 41627 is "Rogers" on Systems **a, b**, and **c**, but is "Gordon" on Systems **d, e**, and **f**).

These issues must be addressed when setting up your system's user ID mapping methodology. Additional information on the types of user ID mapping is presented below.

Transparent User ID Mapping

The *transparent* mode of user ID mapping is specified by the ***n** and ***o** user ID mapping options in the server table. This mode allows incoming calls to retain the same numerical user ID on the called system as on the calling system as long as the numerical user ID appears in a valid */etc/passwd* entry on the called system.

The transparent mode of user ID mapping is intended for use when two or more systems have the same user population and each numerical user ID refers to a single user across the set of systems.

During transparent mapping, it is not possible for the server program to verify that the numerical user ID refers to the same user on both the called and calling systems. Therefore, the user ID is accepted as long as a valid */etc/passwd* file entry – one that has not expired – exists on the called system.

This mode of user ID mapping should only be used when both the called and calling systems have the same user population and user IDs uniquely identify the same user on both systems. If the same numerical user ID (for example, 100) refers to different users on the two systems, do *not* use transparent mapping between those systems: use *translated* mapping instead.

In addition, if transparent user ID mapping is used for administrative logins between two systems, anyone that becomes **root, bin**, or another of the administrative users on one of the systems may retain those privileges on the other system.

Translated User ID Mapping

The *translated* mode of user ID mapping is specified by the **&** user ID mapping option in the server table. The translations are performed using the */etc/opt/dk/dkuidtab* user ID translation file [see *dkuidtab(4)*]. Translated user ID mapping should be used when two or more systems must share some users but the systems do not share common */etc/passwd* files. Numerical user IDs do not need to uniquely identify a single user across the set of systems.

The translated mode of user ID mapping is intended for use when two or more systems may not have the same user population and each numerical user ID may not refer to a single user across the set of systems.

This mode of user ID mapping is significantly more secure than *transparent* user ID mapping since users must authorize themselves with a login and password [see *authorize(1M)*] before using the resources of the called system.

Fixed User ID Mapping

The **fixed** mode of user ID mapping is specified whenever the user ID mapping field of a server table entry lists a specific user name (for example, **nuucp** or **guest**). The specified login must have a valid entry in the */etc/passwd* file. This mapping results in the call using the specified fixed login.

Fixed user ID mapping is often required when invoking system programs that establish sessions such as ***login***. This is shown by the following server table example:

```
201/colan/*!   -    L/t   root  /bin/login  login:-p:-h%H
```

The **authorize** program must read and write files accessible only by **root**, so it must be invoked with a fixed user ID mapping using the **root** login:

```
* authorize /vaex root /opt/dk/sbin/authorize authorize:2O1/58O/1293:%U:%f:%u
```


Fixed user ID mapping is also useful for selecting ***uucp*** logins based on originating location. Each of the logins must exist in the */etc/passwd* file and have */usr/lib/uucp/uucico* as the *login shell*. This allows you to use the */etc/uucp/USERFILE* file to restrict resources based on originating area or access point into the network:

```
201/colan/host*!      uucp   /u   houucp    %s   uucico
201/colan/modem*!     uucp   /u   phuucp    %s   uucico
201/*                 uucp   /u   njuucp    %s   uucico
i1/univ/earth         uucp   /u   nnuucp    %s   uucico
*                     uucp   /u   nuucp     %s   uucico
```

Restrictive User ID Mapping Ranges

The *transparent* and *translated* modes of user ID mapping may be restricted to a numerical range of user IDs by appending the **<uid, >uid, =uid**, or **=uid1-uid2** suffix to the user ID mapping field specifiers. This allows the administrator to restrict mapping to a range of numerical user IDs on a calling host or host group basis.

For example, the administrator may block transparent file transfer access to a range of administrative logins:

```
nj/cc/myux*!  pupu  uR  *n>100  /opt/dk/bin/pupu  pupu:from:%f
```

The administrator may have several systems that are commonly administered, each with a different user population. In this case, transparent mapping could be used for administrative logins and translated mapping for user logins. If administrative logins are all numerically less than 100, the following example shows how three systems (**myuxa, myuxb**, and **myuxc**) may use transparent mapping for user IDs less than 100 and translated mapping for user IDs greater than 100. No other systems have access to the remote login service on this system:

```
# transparent administrative mappings
nj/cc/myuxa     rl   U/vx    *n<100   %s  -Dsh
nj/cc/myuxb     rl   U/vx    *n<100   %s  -Dsh
nj/cc/myuxc     rl   U/vx    *n<100   %s  -Dsh
#
# translated user mappings
nj/cc/myuxa     rl   U/vx    &>100    %s   -Dsh
nj/cc/myuxb     rl   U/vx    &>100    %s   -Dsh
nj/cc/myuxc     rl   U/vx    &>100    %s   -Dsh
```


The mapping entries for the previous example could be shortened to:

```
# transparent administrative mappings
nj/cc/myux[a-c]     rl      U/vx    *n<100  %s              -Dsh
#
# translated user mappings
nj/cc/myux[a-c]     rl      U/vx    &>100    %s             -Dsh
```

Trapping Incoming Calls

The *Server Table Scanning Rules* section describes how the **dkserver** program scans the server table to match and accept incoming calls. Successfully matching a line in the server table results in an accepted call. Failure to find a match in the server table results in a rejected call.

When an incoming call is accepted, a **call answered** code is sent back through the data switch network to the requester. Rejected calls are answered with an appropriate **rejection code** such as *access denied, busy, no answer* or *out of service*. These rejection codes are defined as constants in the header file */usr/include/dkit/dk_unixp.h* with the prefix **Ed**. Examples are shown in Table 3-4.

Table 3-4: Rejection Code Examples

Label	Value	Description
EdBUSY	1	All channels busy
EdNOANSWER	3	Server not answering
EdNOACC	7	Access denied

The optional **T** (trap) server table flag provides a mechanism to reject incoming calls that match a line of the server table. All server table mapping lines that contain the **trap** flag in the flags field will reject an incoming call when the call successfully matches the line. These lines may be considered *trapping lines*. Server table mapping lines that do not contain the **T** trap flag will function as before by accepting the incoming call and generating the specified program according to the other flags in the flags field.

To make the call trapping facility as flexible as possible, the administrator may select the **rejection code** to be used by each trapping line by specifying it as the first argument in the *initial_parameters* field. The remainder of the *initial_parameters* field is parsed for program argument substitution tokens (*special codes*) and is logged to the server table log file as a record of the call rejection.

For example, the following server table line will trap all requests from the originating group **nj/district/trouble** with an access denied rejection code, regardless of the requested service or requesting user ID, and will log a record of the rejection in the server log file:

```
nj/district/trouble  *  T/v adm - 7:TRAPPED(%u from  %f)
```

The above trapping line uses the *wildcard service* to match requests for any service and specifies a fixed user ID mapping of the **adm** login so the line is guaranteed to match all calls from all users in the specified originating group. The program field contains a **null** program specification since no program will be invoked. The arguments field specifies that all calls will be rejected with a rejection code of **7** (access denied; see the header file */usr/include/dkit/dk.h* for a list of other rejection codes) and the requesting user ID and originating group (**%u** and **%f**) will be logged in the server log file for future reference.

The call trapping facility may be used to block "restricted" resources from certain classes of callers as outlined above, or it may be used as a diagnostic aid to help test applications. The following example could be used to test an application's behavior when a required remote resource is busy:

```
univ/product/cust*     appl U  *n>100 /app/appl app:%f:%u:%p
univ/develop/test*     appl T  adm     -      1:BUSY BEHAVIOR TEST
```

In the above example, all requests from production systems for the user defined ***appl*** service are satisfied by invoking the necessary program with appropriate arguments. All requests from test hosts in the development organization are rejected with an ALL CHANNELS BUSY rejection code (rejection code 1) that simulates what would happen if all CommKit Host Interface channels to the called host were in use.

The **trap** flag can be a powerful application testing tool, as well as a barrier to unauthorized host access.

Unauthorized Service Requests

dksrverr can be used to monitor unauthorized requests for CommKit Host Interface services. When **dksrverr** is activated, invalid attempts for services are entered in the *dksrvlog*. (Refer to *dksrverr*(1M) for details.)

For example, for the following *srvtab* entry on **univ/systest/general**:

```
univ/develop/*  login  M  daemon  /opt/dk/sbin/dksrverr dksrverr:brig:%e:%f:%u:grab developer %u
```

will send mail to the **brig** login on **general** whenever anyone from a development system attempts to log onto the system test **general** machine.

Spawning a TLI Application

The **l** flag in a *srvtab* file entry causes **dkserver** to treat incoming calls in the same way as they would be treated by the UNIX Network Listener Service (NLS) ***listener*** daemon process. After appropriate security and user ID checking, the **dkserver** spawns the specified child process. The child process inherits a virtual circuit file descriptor that has a **dktm** driver Transport Provider Interface (TPI) STREAM with the raw data switch virtual circuit STREAM linked under it. Before the new process begins execution, the STREAM is placed into the TPI data transport state, T_DATAXFER, as defined in the system header file *timod.h*.

The **dkserver** process discards the Network Layer Provider Service (NLPS) message that it receives from the call's originating end point. Removing this message from the data stream mimics the behavior shown by the UNIX ***listener*** daemon process. In place of the values found in the NLPS message, the **dkserver** substitutes its own values into the environment for the following variables:

NLSADDR	Set to the originating end point's data switch node address, originating group name, module address, and channel number.
NLSOPT	Set to the null string.
NLSPROVIDER	Set to the value **dktp** with the CommKit Host Interface number appended.
NLSUDATA	Set to the null string.
MPREFIX	Same value as NLSPROVIDER.

The **l** flag is primarily intended to accept requests from other hosts for RFS services without having to create a separate data switch service address and ***listener*** process for that service. However, other TLI or TPI conforming applications can

benefit by using the data switch network together with this CommKit Host Interface feature.

Note: RFS is not supported by the NCR UNIX System.

Typically, the **l** flag is configured on the host advertising one or more mountable RFS file systems to other remote hosts. These remote hosts are set up to call the advertising host's data switch service address with an *rfs* suffix.

The following shows the administrative steps required to set up this configuration. The host that is advertising RFS mountable file systems is referred to as the *RFS server host* and, in this example, has the service address *nj/shore/bird*. The other hosts mounting the RFS server's file systems are referred to as *RFS client hosts.*

To set up the configuration, the UNIX system administrator must execute the following steps:

1. Call the RFS server host from the RFS client host using its data switch service address with an *rfs* suffix. The *rfs* file within the */etc/opt/dk/srvtab* directory must be edited to contain the following entry:

```
*       rfs     /l|timod        listen  /usr/net/servers/rfs/rfsetup    rfsetup
```

Note: The | flag indicates that **dkserver** should push the **timod** transport interface STREAMs module onto the stack before ***rfsetup*** is spawned to begin RFS services.

2. Modify the *rfmaster* file in the directory */etc/rfs/dktp0* on the RFS client host to call the RFS server host:

```
DM1.bird        A       nj/shore/bird.rfs
```

Note: The name **bird** is the RFS server's name in the RFS domain **DM1**.

3. Start up RFS services on the RFS server and RFS client hosts using the ***rfstart*** command.
4. On the RFS server host, advertise the mountable RFS file systems:

```
share -F rfs -o rw -d "bird RFS file sys" /usr/example birdexp
```

5. On the RFS client system, issue a mount request:

```
mount -F rfs DM1.birdexp /mnt
```


Server Table Entries Which Are Not Secure

Since the server table is the absolute arbiter of privilege for services provided by the CommKit Host Interface server program, server table sequences which are not secure must be avoided to prevent unauthorized access to the host system. Administrators should periodically check the server table on each of their hosts and remove any entries which are not secure. This section lists several server table entries which are not secure and describes why they should be avoided. This list is *not* exhaustive; there are other server table entries that can be equally as damaging. The administrator should use these examples as a tool for determining whether a specific server table entry is secure.

The remainder of this section consists of server table entries, each followed by a brief description detailing why the entries should be avoided.

```
*               rl      U/vx        root   %s  -Dsh
*               rx      /vaex       root   %s  -Xsh:-c:%p
```

The previous server table entries are not secure. Both lines allow a requester from any originating system on the data switch network to become the super user on the called system directly. In addition, a normal user (not the super user) on the called system can issue a remote login or execution request to his/her own system (a loop-around request) and also obtain a super user shell.

```
*               rl      U/vx        *n   %s  -Dsh
*               rx      /vaex       *n   %s  -Xsh:-c:%p
```

The above entries do not map all incoming remote login and remote execution requests to super user on the called system, but they will allow the super user of **any** system in the network to become the super user on the called system. Any user on a system with a numerical user ID in common with a user on the called system can access the called system with the permissions of that numerical user ID.

```
nj/yourcc/*     rl      U/vx        *n   %s  -Dsh
```

The above server table entry limits external exposure to systems in the **nj/yourcc** area and exchange. Any super user on a system in that area and exchange can become super user on the called system. A normal user on an originating system with a numerical user ID in common with the called system can access the called system as in the previous example.

Warning: The *n form of user ID mapping should *not* be used in conjunction with originating group wild cards unless those wild cards are sufficiently specific to limit access to trusted, commonly administered systems.

```
*     pupu     -    root     /opt/dk/bin/pupu  pupu:from:%f
```

The specification of **root** on the above server table entry allows any user on any system in the network to read or write any file on the called host. Although this entry does not directly provide a super user shell to the requester, it does allow any requester to replace any file (including */etc/passwd*) on the called system. The **&** user ID mapping specification should be used for **pupu** requests so that users are forced to **authorize** with a login and password before transferring files.

Directory Mode for /etc/opt/dk/srvtab

Server tables that result in reasonably secure host systems generally require fully specified access specifications with limited use of wild cards. Since the server table is searched sequentially from the beginning each time an incoming call is validated, this added security is not without cost. A directory server table format is used to greatly increase call validation performance when a large server table is required.

The server control file (default: */etc/opt/dk/srvtab*) is either a concatenation of all the control files into a single file, or a directory of server control files (there is one for each service provided). Each server table file contains entries for a single service and the file names are the names of the services.

For example, a server table directory might contain files with the following names:

```
-       authorize   do      pupu     rx       whoami
*       dkload      login   rl       uucp
```

Each of the above files with the exception of the file * represents a specific service. When an incoming call request is validated, the server program recognizes that the server table is a directory and attempts to open a file in that directory with a name that matches the requested service (for example, the server would try to open the file */etc/opt/dk/srvtab/pupu* for the file transfer service and the file */etc/opt/dk/srvtab/–* for the **null** service). If it opens, the server scans that file for a

matching line according to the rules outlined in the *Server Table Scanning Rules* section. Lines in each server table file will still match the incoming call request *only* if they contain the service field wild card or if the service field matches the requested service.

If the opening of the appropriate server table file fails, the server attempts to use the file * as the server table file. If the opening of this *catchall* server table file fails, the incoming call request is rejected with an access denied rejection code. Lines in the catchall server table file will only match the incoming call request if they contain the service field wild card or if the service field exactly matches the requested service.

Note: The server will only attempt to use the catchall server table file if the appropriate server table file does not exist or cannot be opened. The server only scans a **single** server table file for each call request.

Additional user-defined services would be provided by adding additional server table files to the server table directory.

Take care when handling the files - and *, since these characters usually have special meaning to editor programs and the shell, [see *sh(1)* in the *UNIX System V Release 4 User's Reference Manual*]. System administrators should use the following commands when editing these files [see *vi(1)* in the *UNIX System V Release 4 User's Reference Manual*]:

```
vi  ./-       # to edit the file "-"
vi  ./\*      # to edit the file "*"
```

Restart **dkserver** when changing to or from a directory mode server control table for the change to take effect; all incoming requests will otherwise be denied.

Summary

Administrators should use these server table facilities to grant remote access to their system for authorized remote systems and users. Wild cards in originating group patterns should be used judiciously, if at all. The administrator should thoroughly examine every entry in his/her server table file.

dkdotab

Use this section as an aid in the editing of */etc/opt/dk/dkdotab*. Refer also to the *dkdotab(4)* manual page.

The CommKit Host Interface software package includes a sample *dkdotab* file found in the directory */etc/opt/dk*. Edit the same file to reflect your own community of interest.

Within a typical network, it is not always possible to support the same or all of a set of commands on all of the hosts in the data switch network. The CommKit Host Interface software allows execution of a given command from a remote host.

The **dkdo** command allows emulation of commands on a host requiring facilities not present on the local host. When executing a command on a local host that does not support the given command, **dkdo** searches the */etc/opt/dk/dkdotab* file for the remote host on which the command should be executed. **dkdo** then places a call through the data switch network to the host that supports the specified command. It sends any input files and arguments to the remote system, executes the command there, and brings back any output files. The input files are placed in a new directory in */tmp* on the remote host, and any output files created as a result of execution of the command are brought back to the local host on the command's completion. The input files in */tmp* are removed after the command is executed. The function of the **dkdo** command is controlled by a **dkdotab** control table that, unless specified, defaults to the */etc/opt/dk/dkdotab* file. This file describes where commands are to be executed and the format of the arguments passed to these commands.

Each **dkdotab** table entry is presented as follows:

```
command   system   flags   options   files
```

The table consists of one or more lines, each with five tab-separated fields as shown above. Each time the command **dkdo** is executed, it scans this *dkdotab* file for the first match with the command requested. If a match is found, the remaining fields indicate:

system the host on which the command should be executed

flags the flags affecting the operation of the **dkdo** command

options the command options available for this command and how the options are parsed

files whether the files associated with this command are input or output files, or both.

The */etc/opt/dk/dkdotab* may consist of one or more lines; one for each unique table entry. The **dkdo** command scans this table for the requested command, looking for the appropriate system on which to execute it. The *command* field holds the name of the command to be executed. The *system* field is the system on which that command should be executed.

The *flags* field may contain several flags to direct the remote command execution. The **s** flag indicates that the file names specified as input are prefixed with "**s.**", and, therefore, the prefix has to be stripped off. The **x** flag specifies that the **dkdo** program should request the ***rx*** service on the remote system rather than the **do** service. Requesting the ***rx*** service executes the user's *.profile* on that system before the remote command. This is necessary when executing remote commands that have special PATH or environment [see *env(1)*] needs. The standard **do** service does not execute the user's remote *.profile* script.

The *options* that are supported for the *commands* entered in the table are all options that follow the conventional syntax [for example, most option syntaxes (such as **-opt arg** and **-optarg**) are allowed, but unconventional syntaxes are not]. The delimiters "**:**" "<" ">" tell the */opt/dk/bin/dkdo* program how to parse the command options. The values in the *options* field are delimited by four possible operators (described in the *dkdotab*(4) manual page).

The *files* field is a description of the type of files that follow the *options* associated with the *command* specified. If there is a colon (:) present in an entry in this field, the flag that precedes it should be inserted in the command line before the command is invoked. This is done after the command line has already been parsed.

Following is an example of an entry in the **dkdotab**:

```
get  fish  sx  r:c:i:x:a:   >*
```

This example means that if the command ***get*** is specified, it should be executed on the host **fish**. If an **s.** precedes the input argument file names, strip off the **s.**

before sending the command arguments to the remote system. Use the ***.rx*** service. If any of the following options are specified, parse them in the following manner (the arguments that follow these option flags are arguments associated with that option whether they follow directly or indirectly after the option key letter): **r**, **c**, **i**, **x**, **a**.

The meanings of these option key letters can be found in the manual page for the respective command. The files associated with the execution of this command are output files (>*).

dkuidtab

Refer also to the *dkuidtab(4)* manual page.

The file *dkuidtab* is found in the */etc/opt/dk* directory. The file is updated by **authorize** when a user from a remote host calls the local host for authorization service, and is used by the **dkserver** program to obtain information on how to map user IDs from incoming call requests to valid IDs on the local host.

The file consists of one or more lines and each line has three fields separated by blanks:

```
originating-group  local-login    local-password
```

where,

- *originating-group* consists of **area/exchange/host.user[/key]**. The **area/exchange/host** is the originating data switch group of the remote host. **user** is the user ID of the user on the remote host. The **key** contains the value of the DKKEY environment variable (if any) on the remote host at the time of authorization.

- *local-login* is the login name of the user on the local host.
- *local-password* is the encrypted password of the user on the local host, or "**:**" if the user does not have a password.

4 Administration

Introduction

This chapter contains the following:

- Notes on maintaining the CommKit Host Interface software
- Procedures for setting up ***uucp*** and **RFS**
- Procedures for setting up and using the TLI
- Procedures for changing the hardware configuration
- Procedures for administering the number of channels
- Procedures for troubleshooting interface problems
- Procedures for running diagnostics on the interface
- Error messages that may be seen on your computer
- Procedures for connecting, administering, and troubleshooting a printer on the data switch network.

Note: RFS is not supported by the NCR UNIX System.

Administrative Notes

UNIX System V Release 4

The UNIX System V Release 4 has several new features that influenced the design of the current CommKit Host Interface software release. This document does not review UNIX System V Release 4, but you should be aware of the new features for this release listed in Table 4-1.

Note: For additional information, refer to the documentation provided with UNIX System V Release 4.

Table 4-1: UNIX System V Release 4 – New Features

New Expanded Features	Description
Directory restructuring	In order to better support diskless workstations, the contents of the directories */etc, /usr, /usr/spool, /usr/bin, /bin, /lib, /usr/lib,* and many others, have been modified. Many of the directories used by CommKit Host Interface software and the *lp* software have been moved or changed in UNIX System V Release 4. You will have to be aware of these changes to configure the network and printers.
Symbolic links	A symbolic link is a file that consists only of a reference to another file. When a symbolic link is accessed, the referenced file is the one that is actually accessed. A symbolic link differs functionally from a regular link in a number of ways: • Files from different file systems may be linked together. • Directories as well as regular files may be symbolically linked. • A symbolic link can be created even if the named file does not exist. • Operations from *sh* such as ***cd ..*** may not work as expected. Symbolic links are used to solve a problem caused by directory restructuring. Existing applications and software (and users) expect UNIX system files and commands to be in standard places. In the cases where directory restructuring has caused a standard file or command to move to a new directory, symbolic links are used to link the old name to the new location.
STREAMS Subsystem	The UNIX system STREAMS feature has been heavily modified and all tty-based drivers have been converted to use the STREAMS architecture. Several of the CommKit Host Interface device drivers have also been converted to use STREAMS features. This means that some aspects of device configuration and troubleshooting will change as well.
Long File Names	The UFS file system type supports file names up to 255 characters long.

As a result of directory restructuring and other design decisions, many of the files and commands have changed in the current release of the CommKit Host Interface software as compared to earlier releases. In general, the directory */etc/opt/dk* contains subdirectories that hold most of the CommKit Host Interface software files and commands. Table 4-2 below will help you locate files in the

current CommKit Host Interface software release.

Table 4-2: File Location Changes

File/Command	Old Location	New Location
User Commands	*/usr/lbin*	*/opt/dk/bin*
Administration Commands	*/etc*	*/opt/dk/sbin*
Host File	*/etc/dkhosts*	*/etc/opt/dk/dkhosts*
Server Control File	*/etc/dksrvtab*	*/etc/opt/dk/srvtab/* (see Note)
Server user ID Mapping	*/etc/dkuidtab*	*/etc/opt/dk/dkuidtab*
dkdo Table	*/usr/lib/dkdotab*	*/etc/opt/dk/dkdotab*
Log Files	*/usr/adm*	*/var/opt/dk/log*
Server Lock Files	*/usr/spool/locks*	*/var/opt/dk/log*

Note: In the current CommKit Host Interface software release, **srvtab** is a collection of files in the directory */etc/opt/dk/srvtab*. Each file contains **srvtab** entries for the service indicated by the file name. Refer to the *srvtab(4)* manual page for additional information.

Files That Grow

The following log files will grow continually unless they are periodically cleaned out by the system administrator.

/var/opt/dk/log/dksrvlog a log of the server activity [see *dksrvlog(4)*]
/var/opt/dk/log/dkuidlog a log of all authorization attempts [see *authorize(1M)*]
/var/opt/dk/log/dkdaemonlog a log of dkdaemon operations [see *dkdaemon(1M)*]
/var/opt/dk/log/dkacct a log of data transfer and connection time [see *dkdaemon(1M)*]

Once a log file reaches the UNIX system maximum file size limit [see *ulimit(2)* in the *UNIX Programmer's Reference Manual*], no further log entries will be appended to that file. It is suggested that the daily task of cleaning out the server log files be put into the administrative **cron** table to be run during off-hours. An alternative for a frequently rebooted system is to clean out the logs at every system reboot by adding appropriate commands to the script */etc/init.d/dkitrc*. The daily server log may be saved in another file and inspected for messages containing the tokens ERROR, DENIED, TRAPPED, or FAILED. See *dksrvlog(4)* .

dkitrc Script File

The script */etc/init.d/dkitrc* is invoked during all init state transitions with an argument of either **start** or **stop**, depending upon which init state is being entered. Links to the file */etc/init.d/dkitrc* in directories */etc/rc0.d* and */etc/rc2.d* determine which argument will be passed to the script during invocation. See *dkitrc(1M)* for the names of these additional links and a more detailed description of */etc/init.d/dkitrc*.

Once the CommKit Host Interface installation has been completed successfully, */etc/init.d/dkitrc* may be edited to automate the starting and termination of additional CommKit Host Interface services or to customize the operation of the default server.

The */etc/init.d/dkitrc* script also controls various CommKit Host Interface functions when the **dkdaemon** process is started. You can modify this script to modify your computer system number of channels, logging or accounting records. The following display is the part of the code that starts the **dkdaemon** process:

```
# Lines are needed here to start dkdaemon either with defaults
# by specifying 'dkdaemon', or with options specified.
# Note that if you start a dkdaemon process with an explicit
# -i 0 for your first interface, you will need to provide
# other daemon(s) with -i int_num, where "int_num" is the interface
# number of the other interface(s) if they exist.
#
# The other daemons can have different -c, -a, etc.
# options, but should not have the -x and -t options.
# For example, to start two daemons with different channel
# number specifications, the following lines can be used:
#      /opt/dk/sbin/dkdaemon -i 0 -c 256 -x -t
#      /opt/dk/sbin/dkdaemon -i 1 -c 128
```

```
# Note that when specifying the -c option, the number of channels
# must agree with the value set in the Data Switch.
#
# Note that when specifying the -c option, the number of channels
# should agree with the -c option on the dkdevs command line above.
#
# By specifying just 'dkdaemon' a single dkdaemon process
# will service all interfaces and the x and t functions.
# See the dkdaemon(1M) man page for more details.

/opt/dk/sbin/dkdaemon
if [ $? -eq 0 ]
then
       echo "dkdaemon process started"
fi
```

The operation of servers is controlled at two places in the script */etc/init.d/dkitrc*. The commands which start servers appear after the comment `#start a dkserver process`, and the commands to terminate servers follow the comment `#stop a dkserver process`.

The following display is the part of the code that starts and stops the **dkserver** process:

```
# start a dkserver process
# add lines here to start dkserver automatically
# for example: 'dkserver' to take all defaults or
# 'dkserver -v 8 -i 1 -s server1 -l /var/opt/dk/log/dksrvlog1'
# See the dkserver(1M) man page for more details

PATH="${PATH}:/opt/dk/bin" /opt/dk/sbin/dkserver
if [ $? -eq 0 ]
then
        echo "dkserver process attempting to start"
fi
;;

stop)
# stop a dkserver process
/opt/dk/sbin/dkserver -t > /dev/null 2>&1
```

```
if [ $? -eq 0 ]
then
        echo "dkserver process stopped"
fi
```

The comments in the above code show the service name with the **-s** option. If no service name is specified, it will be taken from the **uname** of the host. In either case, this name must match the mnemonic name entered in the data switch control computer database. The line that starts the server specifies that */opt/dk/bin* be added to the default PATH variable. The PATH variable is used by the server when starting processes requested by incoming calls. If the PATH variable is not specified, the server will use the value defined in its environment. There are additional options with which the server may be started.

If you are installing more than one CommKit Host Interface board, you will have to start a server in the */etc/init.d/dkitrc* script for each board. The default starts the server for the first interface (interface 0). To start the server for an additional interface, add a **dkserver** line with the **-iX** option (where **X**=1 for the second interface) to the */etc/init.d/dkitrc* script.

Refer to the manual page *dkserver(1M).* You can edit this script to save any of the log files (*dksrvlog*, *dkuidlog, dkdaemonlog,* and *dkacct*).

dkitrc automatically reserves the channel groups defined in *dkgroups*(4) with the following code:

```
# Reserve any groups defined in /etc/opt/dk/dkgroups.
# for group in 'grep -v '^#' /etc/opt/dk/dkgroups | cut -f1'
do
        /opt/dk/sbin/dkmaint -g $group
done
```

Refer to the *dkmaint*(1M) and *dkgroups*(4) manual pages for more details on channel groups.

Linking of Host Interface Files

We suggest you link */opt/dk/bin/dk* to the names of other hosts and other commonly-used destinations listed in the file */etc/opt/dk/dkhosts* [see *ln(1)* in the *UNIX System V User's Reference Manual*]. For example, if the */etc/opt/dk/dkhosts* file contains an entry:

```
fish    nj/shore/fish
```

and you link */opt/dk/bin/dk* to **fish** as follows:

Note: In UNIX System V Release 4, symbolic links can exist across file systems (see ***ln -s*** option.)

```
$ ln dk fish
```

Then the following two commands will be equivalent:

```
$ fish
$ dk fish
```

When using the **dkdo** command to provide distributed ***lp*** or other networked services, it is suggested that the commands you list in the */etc/opt/dk/dkdotab* be linked to the command */opt/dk/bin/dkdo*. This will allow the user to enter:

```
$ command
```

(where *command* is the command to be executed) and have the command execute remotely on the host defined in the */etc/opt/dk/dkdotab* for the command without explicitly entering:

```
$ dkdo command
```

Refer to the *Printer Administration* section for more information on using **dkdo** to provide network printer services.

Special Device Files

The CommKit Host Interface for UNIX System V Release 4 uses a new device naming and numbering convention for the "special files" (or special device files) in the */dev* directory. Each special device file contains a major and minor device number. For more than one CommKit Host Interface, the special device files contain the same major device number but different minor device numbers.

The following are the directories that contain the special device files:

/dev/dk a directory containing raw *dkhs* special device files

/dev/dkt a directory containing *dkty* special device files

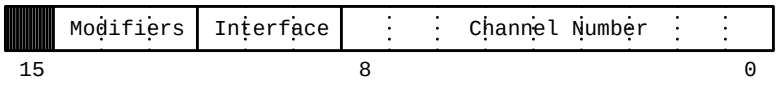
/dev/dkx a directory containing *dkxqt* special device files

/dev/dknp a directory containing *dktli* special device files.

Most CommKit Host Interface special device files are of the format **x.yyy**, where **x** indicates the interface number and **yyy** the channel number.

In UNIX System V Release 4, there can be up to 262144 minor device numbers per device (using 18 bits; the CommKit Host Interface product uses 16 of the available 18 bits). This is an increase over the limit of 255 minor device numbers in earlier releases. Minor device numbers used by the **dkhs** driver are comprised of several bit fields that describe characteristics of the represented device. Each 16-bit minor device number is partitioned as shown in Figure 4-1.

Figure 4-1: Device Number Partitioning (16 bits)



Channel number is nine bits and can represent 512 channels per interface. (Only 256 are used.)
Interface can address eight different physical interfaces but two are currently supported.
Modifier bits are used to distinguish **clone** (dialer) and other special devices from normal data channels.

The minor device numbering scheme and all bit definitions are fully specified in the */usr/include/dkit/devices.h* header file supplied with the software.

The *dkhs* special device files reside in the */dev/dk* directory. Table 4-3 shows examples of the special files of a host having two CommKit Host Interface boards installed.

Note: The special device files differ depending on the number of interfaces supported in the current release.

Table 4-3: Special Device Files – Example – Two Interface Boards

Minor Device Number	Description	Special Files
0	Diagnostic channel for interface 0	*/dev/dk/diag0*
1	Control channel for interface 0	*/dev/dk/ct10*
1-255	Data channels for interface 0	*/dev/dk/0.001*, etc
512	Diagnostic channel for interface 1	*/dev/dk/diag1*
513	Control channel for interface 1	*/dev/dk/ctl1*
513-767	Data channels for interface 1	*/dev/dk/1.001*, etc
4096	Dialer channel for interface 0	*/dev/dk/dial0*
4608	Dialer channel for interface 1	*/dev/dk/dial1*
8192	Default dialer channel	*/dev/dk/dial*

The minor device numbering scheme used by the **dkhs** driver provides for addressing up to 512 data channels on each of eight physical interface boards. The current release of the CommKit Host Interface software supports up to two physical interfaces and 256 channels per interface. The remaining addressability is reserved for future expansion.

The special files for the *dkty* and *dkxqt* drivers also follow the **x.yyy** format, but there are no diagnostic, control, or dialer channels for those drivers.

You need not be concerned with creating or monitoring the special device files since they are created automatically as they are needed. The format of these numbers is described here for your information only. See *dkdevs(1M)* for additional information.

Configuring uucp with d or g Protocol

This section describes how to use the UNIX System V BNU with the CommKit Host Interface software over the data switch network.

The three primary BNU services are ***uucp, ct***, and ***cu***. You can configure the CommKit Host Interface to use one or more of the following protocols:

Protocol	Description
g	The **g** protocol is a ***uucp*** protocol intended to work over noisy low-speed phone lines using modems. It is possible, however, to set up ***uucp*** to use **g** protocol over the CommKit Host Interface. Using the **g** protocol, there will be additional overhead for error checking that may not be necessary.
d	The **d** protocol is designed to operate over the error-free data switch network. The **d** protocol is reliable and transfers blocks of data at a high rate of speed.
e	The **e** protocol is designed to operate with TLI to access the CommKit Host Interface. Refer to the section *Using TLI Support* for procedures on how to set up ***uucp*** using TLI.

Note: If either the calling or the called host is not connected to the data switch network through the CommKit Host Interface software, both ends must use the less efficient **g** protocol.

Configuring Tables for Originating Calls

The ***uucp*** calls may be made to hosts attached to the data switch network by means of a CommKit Host Interface, an RS-232 line (TY12 port), or through a dialer. To accommodate these hardware configurations, the following entries are needed in the originating host's */etc/uucp/Sysfiles*, */etc/uucp/Devices.cu*, */etc/uucp/Devices.cico*, and */etc/uucp/Systems* files.

Note: The */etc/uucp/Devices.cu* and */etc/uucp/Devices.cico* files may have to be created since they are not distributed with the BNU software. The mode, owner, and group should be the same as the */etc/uucp/Devices* file.

1. For calls placed to remote hosts using **d** protocol:

Sysfiles:

```
service=cu      devices=Devices.cu:Devices
service=uucico  devices=Devices.cico:Devices
```

Devices.cu:

```
DKD,d       DKd   0       Any      DK       \D
```

Devices.cico:

```
DKD,d       DKd   0       Any      DK       \D.uucp
```

Systems:

```
host  Any  DKD,d  0  dialstring
```

dialstring is the data switch address of the remote *host*.

2. For calls placed to remote hosts using **g** protocol:

Sysfiles:

```
service=cu      devices=Devices.cu:Devices
service=uucico  devices=Devices.cico:Devices
```

Devices.cu and *Devices.cico*:

```
DK,g        DKg   0       Any     DK      \D
DK          DK    0       Any     DK      \D
```

Systems:

```
host  Any  DK,g  0  dialstring in:--in: login word: passwd
host  Any  DK  0  dialstring in:--in: login word: passwd
```

- *host* is the name of the remote host
- *dialstring* is the data switch address of the RS-232 port on the node
- *login* is the user ID of ***uucp*** on the remote host
- *passwd* is the password of the remote ***uucp*** login.

Note: The **g** protocol can be used by ***uucp*** over TLI. Refer to the section *Using TLI Support* for information.

3. For calls placed to remote hosts using a dialer attached to the data switch node using the **g** protocol:

Sysfiles:

```
service=cu      devices=Devices.cu:Devices
service=uucico  devices=Devices.cico:Devices
```

Devices:

```
ACU   -   0   Any   DK   dialstring   dialername \D
```

- *dialstring* is the service name of the dialer on the data switch node
- *dialername* is the name of the dialer as defined in the *Dialers* file.

Note: The ACU entries need not go into the files *Devices.cu* and *Devices.cico* since they are the same for both BNU services.

Systems:

```
host Any  ACU  baud tel#  in:--in:    login   word: passwd
```

- *host* is the name of the remote host
- *baud* is the baud rate of the ACU
- *tel#* is the telephone number of the remote host
- *login* is the user ID of ***uucp*** on the remote host
- *passwd* is the password of the remote ***uucp*** login.

Note: For calls made to hosts attached to the data switch node by means of an RS-232 line (TY12 port), or calls placed through a dialer, the remote TY12 port must have flow control by device and by the data switch node turned off for ***uucp*** traffic. Otherwise, failures will occur in the transmission of files or in ***Uutry*** attempts.

Configuring Tables for Receiving Calls

This section describes configuration of the host interface tables so ***uucp*** calls can be received through the data switch network on a host equipped with the CommKit Host Interface software.

The **dkserver** program may receive ***uucp*** service requests in the form of requests for the ***uucp*** service. **srvtab** entries of the following form are required to satisfy requests for the ***uucp*** service:

```
area/exch/*!    uucp      u   luucp    %s          uucico
*!              uucp      u   nuucp    %s          uucico
```

More information on the above server table entries can be found in the section *Fixed User ID Mapping* in Chapter 3.

Using TLI Support

TLI support allows TLI applications to operate over the the AT&T data switch network. For some TLI applications, such as ***uucp*** and **RFS**, you must configure the standard UNIX system listener [see *listener(1M)* in the *UNIX System V System Administrator's Reference Manual*]. For RFS service you may alternatively configure your system to use *dkserver*(1M) instead of *listener*(1M). Refer to the section *Spawning a TLI Application* in Chapter 3.

This section discusses how to configure the UNIX system listener for **RFS** and ***uucp***. If you want to configure other TLI applications, refer to the documentation provided with those applications.

The TLI software supports the "connection oriented" and "connection oriented with orderly release" transport services of TLI. Refer to the *UNIX System V Release 4 Programmer's Guide* for information concerning these services.

The Networking Support Utilities (NSU) package must be properly configured before **RFS** and ***uucp*** can be used.

The following sections describe how to:

- Configure the *netconfig* file. This file contains configuration information for the listener and other TLI applications that use the Network Selection Facility.
- Configure the listener. Although some TLI applications do not require a listener, ***uucp*** and **RFS** do. Refer to the documentation delivered with your application for information concerning the need for a listener.
- Start the listener.
- Stop the listener.
- Verify the listener.
- Manually restart the port monitor and listener service.
- Configure multiple listeners.
- Configure multiple TLI Interfaces.
- Configure TLI applications.

For more information on setting up ***uucp***, **RFS**, or a ***listener***, refer to the *UNIX System V Release 4 Network User's and Administrator's Guide*.

Note: RFS is not supported by the NCR UNIX System.

Configure netconfig File

To configure the */etc/netconfig* file, add the following to the file:

```
# netid  semantics     flags   family   proto  device      lookups
dktp0   tpi_cots_ord   -      datakit    -    /dev/dktp0  /usr/lib/straddr.so
```

This creates information for the *netid* or *net_spec dktp0*, with the device file name for access to */dev/dktp0*. Subsequently, the *netid* will be referred to as a *net_spec*.

Configure a Listener

A standard NSU program, ***listener***, is used to listen for service requests on the network. The listener is the network server for each host and must be properly configured for applications, such as RFS, to work.

Your computer may be connected to more than one network (for example, data switch and StarLAN). Each network *must* have its own listener. The default device name (*net_spec*) of the CommKit Host Interface network provider is *dktp0*, and the **clone** device, which provides access to the network, is */dev/dktp0*. See *dktli(7)* for more information.

To configure the listener:

1. Verify that the transport device for your interface exists as defined in the *netconfig* file. The default for interface 0 is */dev/dktp0* [see *dktli(7)*].

2. Verify that the listener's address (service name) has been configured on the data switch. This is done by using the ***verify address*** command on the node console. The listener's address is specified by the **-l** option of the ***nlsadmin*** command. The listener's address must be different than the one used by **dkserver**, otherwise the listener will fail to start. If the service name has not been configured, use the ***enter address*** command on the data switch node console to enter the host name in upper case characters.

   Note: The listener's address is usually the host name in upper case characters [see *uname(1M)* in the *UNIX System V System Administrator's Reference Manual*] since the lower case host name is already used as the service name by the **dkserver** program. If the service name of your system does not contain any alphabetical characters, you must select a new service name for the listener, since the listener and the **dkserver** program would have the same service name.

3. Configure the listener by using the ***sacadm*** and ***nlsadmin*** commands.

   The following is an example of configuring a listener for interface 0 with an address of *nj/shore/GULL*. The listener uses the *net_spec* name of **dktp0**, area **nj**, exchange **shore**, and the host name **bird**.

Note: In this example the listener name, GULL, appears in upper case.

```
# sacadm -a -p dktp0 -t listen -c "/usr/lib/saf/listen dktp0" \
 -v `nlsadmin -V` -n10 -y"datakit GULL"

# nlsadmin -a0 -c "/usr/lib/saf/nlps_server" -y "NLPS server" -w root dktp0

# nlsadmin -l "nj/shore/GULL" dktp0
```

Note: UNIX System V Release 4 Service Access Facility (SAF) will automatically start and stop the listener on init state change. Refer to *sacadm(1M)* and *nlsadmin(1M)* in the *UNIX System V Network Administrator's Reference Manual* for more information.

Starting the Listener

The standard SAF starts all defined listeners on transitions to init state 2. It is normally unnecessary to start the listener manually. However, the listener for the **dktp0** network can be started manually with the following command:

```
# sacadm -s -p dktp0
```

See *sacadm(1M)* and *nlsadmin(1M)* in the *UNIX System V Network Administrator's Reference Manual*.

Stopping the Listener

The standard SAF stops the listeners for all *net_specs* on transitions to single-user mode. Usually it is unnecessary to stop the listener manually. The listener for *net_spec* **dktp0** can be stopped with the command:

```
# sacadm -k -p dktp0
```

Verifying the Listener

To verify a listener has been configured properly follow the procedures in this section. Display the status of the listener on *net_spec dktp0* by entering:

```
# sacadm -l -p dktp0
```

If the status is ENABLED, the listener has been configured properly.

If the status is STARTING, repeat the ***sacadm*** command every 30 seconds until the system displays ENABLED, INACTIVE, or FAILED.

If the status is INACTIVE or FAILED:

1. Check the listener log file for error information.
2. If the listener cannot open the device, verify that the device is correct and **dkdaemon** had been started with the option **-t**.
3. Because it may have terminated, verify that the **dkdaemon** is active. Examine the file */var/opt/dk/log/dkdaemonlog* for errors reported by **dkdaemon** for the component **dknp**. If this component failed to start, verify that the file */dev/dknp/daemon* exists. See *dkdaemon(1M)*.
4. Verify the *netconfig* file; refer to the section *Configure netconfig File*, earlier in this chapter. Refer also to *netconfig(1M)* in the *UNIX System V Network Administrator's Reference Manual*.
5. Verify the listener address on the data switch is restored to service.
6. Manually start the listener; refer to the section *Starting the Listener*, earlier in this chapter.
7. Display the listener configuration by entering:

```
# sacadm -l -p dktp0
```

8. If the status is still INACTIVE, call your support hotline.

Note: The *listener* must be ENABLED before it will accept incoming calls.

Manually Restart the Port Monitor and Listener Service

In the case of reinstallation of CommKit Host Interface software, once the previous version of the software is removed, the */dev/dktp/***X** devices (**X** is the interface number) which were previously defined in */etc/netconfig* will be removed. If **rpcbind**(1M) attempts to open these non-existent devices, the following error message will be generated for each interface:

```
/dev/dktpX cannot open connection: System error
```

This message indicates the *dktp* devices have not had a chance to be created after the software was reinstalled. The TLI server processes will appear to be ACTIVE, but will fail as soon as they are accessed because **rpcbind** has not successfully bound the server to an address.

For this reason, or at any time the listener process(es) fail, the administrator must start and stop the port monitor and network listener service of the SAF hierarchy by using the ***sacadm*** command:

```
# sacadm -k -p dktpX

# sacadm -s -p dktpX
```


Configure Multiple Listeners

TLI support is initially configured for a single listener with a *net_spec* (network specification) of **dktp0**. It is possible, however, to configure multiple listeners. The maximum number of listeners permitted is limited by the number of *servers* supported per interface by the data switch (see the appropriate AT&T *Data Switch Administration Guide*).

Multiple listeners can be used to make some listener services public (for example, ***uucp***) while keeping others private (for example, RFS and private applications), or when the incoming call load must be shared between multiple interfaces.

sacadm requires each listener on a system have a unique *net_spec*. Since the **dktp0** *net_spec* is used by the listener distributed with the software, a new and unique *net_spec* must be assigned to each new listener configured. For example, an additional listener on interface 0 may be configured with a *net_spec* of **dktp0a**.

A new */dev* entry must be created for each additional listener by making links [see *ln(1)* in the *UNIX System V User's Reference Manual*] to the appropriate */dev/dktp**X*** entry. The following is an example of creating the necessary */dev* entry for the new **dktp0a** listener:

```
# ln /dev/dktp0 /dev/dktp0a
```

Once the */dev* entry has been created for the new **dktp0a** listener you must do the following:

1. Configure the *netconfig* file for **dktp0a**; refer to the section *Configure netconfig File*
2. Configure the listener for **dktp0a**. You must use a unique service name, refer to the section *Configure a Listener*
3. Verifying the listener; refer to the section *Verifying the Listener*.

Configure Multiple TLI Interfaces

TLI supports access to each physical interface installed on your system. To configure additional interfaces to operate with TLI:

1. Verify that the special device files */dev/dktp**X*** (where ***X*** = 0 or 1) exist by entering:

```
# ls -l /dev/dktpX
```

If the devices exist, go to step 3. If the */dev/dktp**X*** files do not exist, go to step 2.

2. To create the necessary special device files, use the following command:

```
# /opt/dk/sbin/dkdevs -Mmaj_list
```

Where *maj_list* is a comma separated list of major number with no spaces between entries. The major numbers to include in *maj_list* are obtained by executing **getmajor 776**.

3. Configure *netconfig* file; refer to the section *Configure netconfig File.* Add entries for additional interfaces in which you substitute **dktp***X* for **dktp0**.
4. Configure the listener. You must use a unique service name, refer to the section *Configure a Listener*. Add entries for additional interfaces in which you substitute **dktp***X* for **dktp0**.
5. Verify the listener; refer to the section *Verifying the Listener*. Substitute **dktp***X* for **dktp0**.

Configuring RFS to Use TLI

Note: RFS is not supported by the NCR UNIX System.

Refer to the *UNIX System V Release 4 Network User's and System Administrator's Guide* for additional information on configuring RFS.

After installing the CommKit Host Interface software on your computer system, configure RFS using the **root** login:

1. The RFS utilities package must be installed on your system. If this package is not installed, load it from its media.

2. Verify that the listener has been configured with its full network address. Failing to properly configure the listener network address will result in difficulties when attempting to start RFS. The current network address for the **dktp0** listener may be printed with the following command:

```
# nlsadmin -l - dktp0
```

Note: Network device is **dktp0** and the address should be the full network address. For the data switch, the full network address must be used where the RFS documentation calls for the address.

3. Follow the procedures in *UNIX System V Release 4 Network User's and System Administrator's Guide* to set up RFS.
4. Verify that the RFS service code 105 has been configured by entering:

```
# nlsadmin -v dktp0
```

5. If the RFS service code 105 has not been configured, use the ***nlsadmin*** command to configure the RFS service.

```
# nlsadmin -a105 -c"/usr/net/servers/rfs/rfsetup" -y "RFS server" dktp0
```

6. Start RFS by one of the following methods:
   - If the host is the primary name server, enter:

```
# rfstart
```

- If the host is not a primary name server, enter the following, where **network_address** is the full network listener address of the primary name server:

```
# rfstart -p network_address
```

Configuring uucp to Use TLI

This section describes how to use the UNIX System V BNU, which contains ***uucp***, to run over TLI. ***uucp*** can be configured to run over TLI using either the **g** or **e** protocol; TLI does not support the **d** protocol.

Note: The **cu** and **ct** cannot run over TLI unless a suitable TLI Terminal Interface package is available. The CommKit Host Interface software does not provide a TLI Terminal Interface package. The standard BNU software can be used over the TLI support.

Protocol for BNU Over CommKit Host Interface Channels

The standard protocol used by ***uucp*** (**g** protocol) is designed for communications over telephone lines.

The **e** uucp protocol is designed for use with error-free communications channels using Transport Library Services. The **e** protocol uses the TLI to access the CommKit Host Interface software STREAMS drivers for connections to the data switch network [see *dktli(7)*]. This protocol may only be used when both the calling (local) and the called (remote) hosts are connected to the data switch network by means of a TLI-compatible version of the host interface.

If either the calling or the called host is not connected to the data switch network through the host interface, both ends **must** use the less efficient **g** protocol.

Configuring Tables for Originating Calls

The **e** or **g** protocol may be used to call any remote host that is connected to the data switch network by means of a TLI-compatible version of the host interface. Originating calls using the **e** protocol access the host interface by means of the TLI-compatible network provider. The following entries are necessary to support originating calls using the **e** or **g** protocol:

Devconfig:

```
service=uucico   device=NP        push=tirdwr
```

Devices.cico:

```
NP,eg    dktp0    -        -        TLIS     \D  nlsnp
```

Dialers:

```
nlsnp    ""       ""       NLPS:000:001:101\N\c
```

Systems:

```
host  Any  NP  -  area/exchange/ADDRESS
```

where *area/exchange/ADDRESS* is the full network address of the listener running on the called system *host*.

For example, if your system **fish** wants to be able to communicate with system **bird** the entry on the system **fish** would be as follows:

```
bird    Any    NP     -     nj/shore/GULL
```

If you wish to use ***uucp*** over multiple interfaces, perform the following procedures for each interface:

1. Verify that the other TLI interfaces have been configured as described in the section *Configure Multiple dktli Interfaces*, earlier in this chapter.
2. Modify the following files, where *host* is the remote host, *area/exchange/ADDRESS* is the full network address of the listener running on the remote host, and ***X*** is the interface number.

Devconfig:

```
service=uucico   device=NPX   push=tirdwr
```

Devices.cico:

```
NPX,eg   dktpX   -   -   TLIS    \D   nlsnp
```

Systems:

```
host  Any  NPX   -   area/exchange/ADDRESS
```

For more information on using the BNU with a TLI-compatible network provider, refer to the following files:

/etc/uucp/Sysfiles */etc/uucp/Systems* */etc/uucp/Devconfig*
/etc/uucp/Devices */etc/uucp/Dialers*

Note: These files contain information that will help in understanding TLI configuration.

Refer also to the *UNIX System V Release 4 System Administrator's Reference Manual.*

Configuring Tables for Receiving Calls Using TLI

To configure the necessary CommKit Host Interface tables to allow ***uucp*** calls to be received through the data switch network on a host equipped with the CommKit Host Interface software and TLI Software follow the procedures in this section.

uucp calls can be made to your system through the CommKit Host TLI Interface in the following two ways; each requires a special table entry:

1. The network listener may receive ***uucp*** service requests (listener service code **101**) from other hosts equipped with a TLI-compatible version of the CommKit Host Interface software and must be configured to accept those requests. The listener is configured to accept requests for service code **101** when the CommKit Host Interface software is initially installed. The commands to perform this configuration are:

```
# nlsadmin -auucp -c "/usr/lib/uucp/uucico -r0 -iTLI -unuucp" -yuucp dktp0

# nlsadmin -a101 -c "/usr/lib/uucp/uucico -r0 -iTLI -unuucp" -yuucp dktp0
```

2. The network listener may receive ***login*** service requests which ultimately result in a ***uucp*** user logging in (with a ***login*** and ***passwd***) and running the **/usr/lib/uucp/uucico** program. This scenario requires the installation of TLI-compatible ***login*** service software that is not part of the standard CommKit Host Interface software distribution.

Other TLI Applications

To configure other TLI applications on your computer, use the documentation provided with that application and review the configuration sections for RFS and ***uucp***.

When configuring other TLI applications remember:

- The *net_spec* is **dktp0, dktp1**, and so on for interface 0, 1, etc.
- The full network address must be used where system addresses are required. Refer to the section *Configuring uucp to Use TLI*. For example, the ***nlsadmin*** command would use **area/exchange/NODE** for the **-l** option.
- TLI support is composed of a multiplexer, **dktm** [see *dktli(7)*].
- The CommKit Host Interface software does not support connectionless TLI applications.

Changing the Hardware Configuration After the Initial Installation

This section describes how to add or remove host interface boards.

To change the number of interfaces on your computer:

1. Log in as **root.**
2. Shut down the system:

```
# cd /
# shutdown -y -g0 -i0
```

3. Install or remove an interface board according to the procedure described in Chapter 2.
4. Run diagnostics on the interface board. Refer to the section *Diagnostics*.

5. Start the interface. Refer to the section *Verify Operation* in Chapter 2.
6. Modify the */etc/init.d/dkitrc* script. Refer to the section *dkitrc Script File.*

Changing the Number of Channels

Note: The number of channels defined on your computer must match the number of channels defined on the data switch node.

To change the number of channels on your computer:

- Add the **-c** option to the line that invokes the **dkdaemon** program in the */etc/init.d/dkitrc* file (default 64). The following example changes the number of channels to 128 for all interfaces in the */etc/init.d/dkitrc* file:

```
/opt/dk/sbin/dkdaemon -c128
```

You must modify or add the **-c** option to the line that invokes the **dkdevs** program in the */etc/init.d/dkitrc* file (default 64). The following example changes the number to 128, as in step 1.

Note: You may receive an error message if the number of channels used to start **dkdaemon** does not match the number of channels configured in the data switch database. Use the ***verify cpm*** command to confirm the number of channels configured in the database. The NCHLS field of the ***verify cpm*** command is the value that should match the value of the **-c** option used by **dkdaemon** (refer also to Figure 4-2).

```
/opt/dk/sbin/dkdevs -c128 -Mmaj_list
```

To change the number of channels in the data switch node:

- Enter the following on the data switch node console, where **xx** is the slot number of the CPM-HS module. You will be prompted for additional information.

```
CC0> remove cpm xx
CC0> change cpm xx
CC0> restore cpm xx
```

Note: If sufficient memory is not available on the data switch node, it may be necessary to reboot the node.

Troubleshooting Facilities

The CommKit Host Interface package provides troubleshooting facilities that may be used to verify an interface operation or help isolate problems.

Software Troubleshooting Procedures

The following sequence of steps should be used to locate problems with the CommKit Host Interface software. This sequence should be used after a failure is detected in the execution of a CommKit Host Interface command.

1. Verify that the **dkdaemon** process is running.
2. Verify that the **dkserver** process is running.
3. Verify that the collection of files in */etc/opt/dk/srvtab* exist and are properly configured. Refer to *srvtab(4)*.
4. Examine the server log (*/var/opt/dk/log/dksrvlog*) for ERROR, DENIED, or FAILED messages which indicate a problem with the service requested.
5. Verify that the number of channels defined in the data switch control computer database is consistent with the number of channels defined in the */etc/init.d/dkitrc* file (refer to the section *Changing the Number of Channels After the Initial Installation*). Execute the command

```
CC0> verify cpm XX
```

   (where *XX* is the number of the slot in the data switch node in which the CPM-HS module resides) on the data switch console.

6. Verify that the CommKit Host Interface devices were made correctly by examining the */dev/dk* directory for the device nodes referencing the data switch. For more information, refer to the section *Special Device Files*, earlier in this chapter.
7. Verify that the server and listener names are properly defined by using the ***verify*** commands as shown in Figure 4-2 (using the names assigned in Figure 1-2 as an example).

Figure 4-2: verify Commands

```
CC0> verify cpm XX
   94-05-17 13:08:42 NODE=ocean
M  verify cpm XX
   MODULE ADDRESS: XX
   MODULE TYPE: cpmhs                NCHLS: 256
   SERVICE STATE: in                 SERVICE TYPE: unix
   HOST: bird
   COMMENT:
   BILLING  PERIODIC BILLING
   off      off
   GROUP     CHNLS     CHNL RANGE     EPN          CUG PROFILE
   bird      254         2-255

CC0> verify host bird
   94-05-17 13:08:50 NODE=ocean
M  verify host bird
   HOST: bird
   CURRENT GROUP/ADDRESS ASSIGNMENTS:
   GROUP       MNEMONIC     X.121        LEVEL     SERVICE STATE
   bird        bird                      local          in
               GULL                      local          in

CC0> verify group bird address
   94-05-17 13:09:08 NODE=ocean
M  verify group bird address
   GROUP: bird         TYPE: local DEV/HOST: bird
   DIRECTION: 2way     PASSWORD: no
   ROUND ROBIN SERVICE: none
   ADDRESSES:
   MNEMONIC              X.121             LEVEL
   bird                                    local
   GULL                                    local
```

Troubleshooting the Host Interface Communication

If the server does not show ACTIVE and SERVING in the server log (default is */var/opt/dk/log/dksrvlog*) or if the data switch console does not show the CPM-HS module connected to the host to be in the SERVING state, then the following should be executed on the host:

Note: If you have more than one CommKit Host Interface board installed and you are having problems with more than one interface, perform the following procedures for each interface.

1. Stop the server process on the host if the server log does not show ACTIVE and SERVING. If the server log shows ACTIVE and SERVING but the CPM-HS module is not ACTIVE, you do not have to kill the server process.
2. Stop the **dkdaemon** process by using the ***ps*** command to determine the process number of the **dkdaemon** process and terminate it by using the ***kill*** command.
3. Start the **dkdaemon** process by entering the following using the same *options* as you have configured in your *dkitrc* file.

```
# /opt/dk/sbin/dkdaemon options
```

where *options* is a list of options found in */etc/init.d/dkitrc*.

4. Reset the interface by executing the following command:

```
# /opt/dk/sbin/dkmaint -r -iX
```

where *X* is the interface number.

5. Start **dkserver** with the following command by entering the same options as you have configued in your *dkitrc* file:

```
# /opt/dk/sbin/dkserver options
```

6. Enter a ***display connections*** command on the data switch node console to verify that the host is ACTIVE and SERVING.

   Note: It may take two minutes for **dkserver** to recover automatically if it is still running; however, outgoing calls to other hosts should work immediately.

```
CC0> disp conn mod XX
```

   where *XX* is the slot number in the data switch node where the CPM-HS module is installed.

7. If the server still does not start, verify the server's address is in service and execute the following commands on the data switch console:

```
CC0> restore addr <name>
CC0> remove cpm XX
CC0> restore cpm XX
```

<*name*> is the server's address

XX is the slot number of the CPM-HS module.

8. Check that the server log shows ACTIVE and SERVING and the CPM-HS module shows SERVING.
9. If the server still will not start successfully, verify all fiber optic connections between the host transmitter and receiver and the data switch node.

10. Run diagnostics. Refer to the *Diagnostics* section.

If the diagnostics fail, refer to the section *Run Diagnostics on CommKit Host Interface Board* in Chapter 2.

Reading Status and Statistics

The CommKit Host Interface software package provides the user with a function to retrieve status information from the drivers.

The **dkstat** command [found in the directory */opt/dk/sbin* – see *dkstat(1M)*] returns status information maintained by the **dkhs** driver during operation. This information is then formatted and printed on the user's terminal.

The **dkhs** driver currently maintains several cumulative counts associated with the low-level protocol. Examples include the numbers of bad blocks received and retransmitted blocks. A description of the fields can be found in the *dkstat(1M)* manual page.

You must log in as **root** to execute *dkstat(1M).* A sample output is provided below:

```
# /opt/dk/sbin/dkstat

Interface Statistics
---------------------
x_intr    =        31132
x_xhf     =            0

r_intr    =          143
r_nobuf   =            0
r_parity  =            0
r_frame   =            0
r_ovrun   =            0

Channel Summary Statistics
--------------------------
x_msgs   =        30325
x_blocks =        31336
x_bytes  =       410602
```

```
x_enq    =            0
x_init   =           41

r_msgs   =         5638
r_blocks =         3187
r_bytes  =        47599
r_badblk =            0
r_rej    =            0
r_enq    =            2
r_ireq   =            8
r_init   =           36
```

Diagnostics

Two types of diagnostics are available: 3B2 Computer off-line diagnostics and looparound diagnostics run on the data switch control computer.

3B2 Computer Diagnostics

For the 3B2 computer, diagnostics are executed off-line in the same manner as diagnostics for all other 3B2 feature boards.

Two phases are run each time the 3B2 computer is booted. Full diagnostics should be run on new boards or when problems are encountered with the host interface.

To run all phases of diagnostics, shut the system down to the firmware mode, boot the diagnostics monitor, **dgmon**, and enter:

```
DGMON> dgn dkpe x ph=1-15
```

Where *x* specifies the interface (0 or 1) on which you want to run the diagnostics. To run the diagnostics on all interfaces, omit the variable *x*. Refer to the *Diagnostics* section in Chapter 2 for complete details.

Data Switch Control Computer Looparound Diagnostics

Two loop-around tests are available on the data switch control computer:

Local_loop tests the CPM-HS module and CPM-HS paddle board.

Remote_loop tests the CPM-HS module, paddle board, and the fiber optic link.

Local_loop

The local_loop test requires you to remove the module's fiber optic cable from the CPM-HS paddle board and replace it with a loop-around cable. The CPM-HS module must be out of service for this test.

The command ***diag cpm*** enables you to perform the local_loop test. See the following example for a typical local_loop test.

```
CC0> diag cpm
MODULE ADDRESS: 11
TEST TYPE [command_logic, module_reset,
        local_loop, remote_loop: +(local_loop)]: local_loop
REPETITIONS [1-1000; 'c' for continuous: +(1)]: <Enter>

Replace the fiber optic link connection on the paddle board
with a loop-around connector.
This task must be performed within 60 seconds!
Type yes To Continue, no To Stop Command:
CONTINUE TESTING [yes, no: +(yes)]: yes

   diagnose cpm 11 local_loop
   Loop-Around Test Successful
```

Remote_loop

The remote_loop test requires you to remove the fiber optic cable at the host interface board and connect the two cable ends with a looparound connector. The CPM-HS module must be out of service for this test.

The command ***diag cpm*** enables you to perform the remote_loop test. See the following example for a typical remote_loop test:

```
CC0> diag cpm
MODULE ADDRESS: 11
TEST TYPE [command_logic, module_reset,
          local_loop, remote_loop: +(local_loop)]: remote_loop
REPETITIONS [1-1000; 'c' for continuous: +(1)]: <Enter>

Ask the host computer administrator to replace the fiber
optic link connection on the FIB paddle board with a loop-around connector.
This task must be performed within 60 seconds!
Type yes To Continue, no To Stop Command:
CONTINUE TESTING [yes, no: +(yes)]: yes

   diagnose cpm 11 remote_loop
   Loop-Around Test Successful
```

CommKit Host Interface Error Messages

The following sections identify error messages that may be seen on your computer and error messages that users may see when executing some of the commands that make outgoing calls over the CommKit Host Interface.

Console Error Messages

This section describes several error messages that may appear on the console as a result of errors detected in the CommKit Host Interface software and hardware.

Messages that begin with the string:

```
dkhsN:
```

are printed by the **dkhs** driver and refer to the specified interface or unit device. **N** indicates the interface number.

Messages that begin with the string:

```
dkux(N):
```

are printed by the **dkux** STREAMS module and refer to the specified circuit or channel.

The following message may occur during heavy call setup / takedown loads:

```
(DKUXn) Garbage UNIXP Message Received on Interface <n>
        xxxxxxxx xxxxxxxx xxxxxxxx xxxxxxxx
```

This message indicates an unknown control message has been received but has not affected the call processing on the interface. The Garbage message (indicated by *xxxxxxxx*) is printed to allow for analysis of other unusual conditions.

Certain heavy host and node call processing conditions can cause the data switch note to send a control message of the incorrect length. In such a case, the host will display the following:

```
NOTICE: dkux_unixp_parse: wrong length (20)
        xxxxxxxx xxxxxxxx xxxxxxxx xxxxxxxx
        xxxxxxxx
```

to indicate the control message is not of the expected length. The fields (indicated by *xxxxxxx*) following the *NOTICE* line above indicate the actual contents of the incorrect message. The condition is non-fatal because the protocol from the node and the host will resend the message to recover.

Hardware Error Messages

If the CPM-HS module in the data switch node goes out of service after the CommKit Host Interface server is started, if the CPM-HS is removed from service during operation, or if a power failure occurs on the data switch node, an error message will appear on your host console indicating the loss in communication with the data switch node.

```
NOTICE: dkhsN: Interface is down: RX-MUTE
```

The error type RX-MUTE defines a receive error. The message TX-HANG indicates a transmit error condition.

```
NOTICE: dkhsN: Interface is down: REASONS
```

The error indicates the software on the interface board has stopped running because of administrator action or an error condition for a stated REASON.

```
dkhsN: Interface Restarted
```

This message indicates the interface has automatically restarted after a previously detected error.

```
dkpeN: FAULT x%x during sysgen
```

This message indicates that a hardware problem occurred while initializing the interface board. The board is not properly installed or is bad.

```
dkpeN: No response to sysgen
```

This message indicates that the interface board did not respond to an initialization request within the required time interval. The board may be bad or there may be a software configuration error.

```
dkpeN: Problem with Fiber Cable
```

This message usually indicates that the **dkdaemon** process has tried to start the interface software when the CPM module is out of service or the fiber cable is not properly connected.

Software Error Messages

The following error message may be encountered when there is a problem with the remote end point's requested URP transmitter window size.

```
dkux_dial_connect: undersize rbufsz 8 for (0,5), using default (8)
```

This is a notice which indicates the requested window size was unspecified or less than the minimum that the host's URP transmitter can send. The message notifies the user that the virtual circuit may have difficulty during its data transfer phase.

Server Error Messages

When communications between your computer and the CommKit Host Interface node are interrupted, the server detects the problem and prints the following error message on the host console:

```
* * * * * * * * * *
* * * The CommKit Server for servername is having problems
* * * Please get someone to attend to it.
* * * The last error was: date
* * * dkmgr: Unable to contact CommKit for Server servername
* * * * * * * * * *
```

servername is the name of the server reporting the problem. The servername must be properly defined in the data switch node's database.

date is the date, time, process ID, and channel number of the problem. This has the following form:

```
Sep 25 08:45:40 (112) [0.000]
```

The number in parentheses is the process ID of the **dkserver** process encountering the problem. The number in brackets [0.000] is in **x.yyy** format, where **x** indicates the interface number and **yyy** indicates the channel number upon which the error occurred.

To clear this error message, examine the contents of the *dksrvlog* file for additional information and check the state of the CPM-HS module on the data switch

node console and all hardware connections.

When service is restored to the CPM-HS module, the following message will appear on your console:

```
dkhsN: Interface Restarted
```

Outgoing Call Error Messages

The following are error messages that are not necessarily hardware related, but may appear when placing an outgoing call through the CommKit Host Interface. Error codes from the data switch node are interpreted by the CommKit Host Interface software and a descriptive message is printed at the terminal.

For example, consider the following entry and error message:

```
$ dk nj/shore/bird

dkdial(intf 1): Can't connect to nj/shore/bird.rl.vx, dk_errno 4: Destination not recognized
```

This indicates the destination does not exist or the network does not allow the host access to the destination.

The error messages are described below:

- **Access denied**

  The call was denied by the remote server or network security. See *dkserver(1M)*. This error can also occur if this host attempted to set up a server, but the data switch control computer database is not prepared to accept such a setup from this host. This can be caused by:

1. The server name is not defined in the data switch control computer database

2. The name is not assigned to a group
3. The group is not assigned to the host-connected CPM-HS module.

- **Address too long**

  The call was denied because the length of the dialstring was larger than the network maximum.

- **All channels busy**

  All assigned ports/channels are in use or are marked as not available by the remote endpoint.

- **All trunk channels busy**

  One of the network control computers has run out of some resource and is unable to process the call at this time.

- **Auto dialer failed to initiate call. Try again**

  The autodialer on the called port responded to the data switch node and failed during dialing. If this message appears more than twice in succession, contact the data switch Network Administrator.

- **Bad parameter**

  The *dkdial*(3X) routine was called with an invalid parameter.

- **Call did not go through. Try again.**

  The autodialer on the called port responded to the data switch node and failed during dialing. If this message appears more than twice in succession, contact the data switch Network Administrator.

- **Call Failed**

  Unknown call setup or remote host error.

- **Connection broken. Try again later**

  The call cannot be completed. The connection was broken enroute to the destination. Try again.

- **Could not complete your call. Try again**

  The call could not be completed because:

— The autodialer failed to complete the dialing sequence, or

— The port connected to the autodialer was removed from service during the call.

- **Destination not recognized**

  Some part of the requested destination is not defined in the network. The network does not allow this host to have access to the requested destination. The requested destination is not well formed (too many slashes).

- **Dial to vlp error**

  The call could not be completed because an error occurred converting the dialstring to vlp format.

- **Dialed number busy**
  Dialed number is busy

  The call was dialed successfully and a busy signal was detected.

- **Dialer error**

  The call could not be completed due to an error detected by an old autodialer.

- **Dialstring too long**

  The call was denied because the length of the dialstring was larger than the network maximum.

- **Directory Assistance**

  The user has requested directory assistance.

- **Dkserver: Call on a busy device or call collision, try again.**

  The call could not be accepted because the host device is busy or because the incoming call collided with an outgoing call. A busy device is one that is being cleaned up from a prior call or a device that is held open by some lingering process.

  You may also see this error message on an *incoming* call from a `DESTINATION` prompt displayed as error code **136**.

Note: If you try to contact a remote host from the `DESTINATION:` prompt, any one of the errors 130–136 may occur. The error message will be identified by its appropriate error code in the following error message where xxx is the code number. (Refer to the manual page *dkerr(3X)* for more information on error messages codes.)

```
ERROR DURING CALL SETUP. CALL SYSTEM
ADMINISTRATOR. CODE: xxx
```

- **Dkserver: Can't chroot. Call System Administrator**

  The call could not be completed because the remote server could not change **root** to the *home* directory of the caller on the remote host.

  You may also see this error message on an *incoming* call from a `DESTINATION` prompt displayed as error code **134**.

- **Dkserver: Can't open line. Call System Administrator**

  The call could not be completed because the remote host interface server could not open the data switch special device file needed to accept the call. See *dkdevs(1M)*.

  If you are using the **dk** command and this error occurs, it may be due to improper configuration of **dkdaemon** on the machine you are calling. See the *dkdaemon(1M)* manual page for more information on the **-x** option.

  You may also see this error message on an *incoming* call from a `DESTINATION` prompt displayed as error code **130**.

- **Dkserver: Can't push your streams module.**

  The call could not be completed because the configured STREAMS modules could not be pushed onto the open channel.

  You may also see this error message on an *incoming* call from a `DESTINATION` prompt displayed as error code **131**.

- **Dkserver: Can't set/get circuit parameters: Call System Administrator.**

  The call could not be accepted because the **dkserver** process could not communicate with the host device. Either the incoming initialization or buffer size could not be set or the the receiving buffer size could not be

returned in the answer.

You may also see this error message on an *incoming* call from a `DESTINATION` prompt displayed as error code **135**.

- **Dkserver: Invalid protocol requested.**

  The call could not be completed because the remote server did not understand the connection protocol.

  You may also see this error message on an *incoming* call from a `DESTINATION` prompt displayed as error code **132**.

- **Dkserver: srvtab not readable. Call System Administrator**

  The call could not be completed because the server tables (files in */etc/opt/dk/srvtab*) on the remote host were unreadable or damaged.

  You may also see this error message on an *incoming* call from a `DESTINATION` prompt displayed as error code **133**.

- **Endpoint hung up**

  The endpoint hung up the call. This is not an error.

- **Facility not subscribed**

  The facility on the SIM was not subscribed to.

- **Hop count exceeded**

  The call could not be completed because the hop count configured for this node was exceeded.

- **Host configuration mismatch. Call Network Administrator**

  The call was rejected because the host channel picked for the outgoing call is not configured on the network. Contact the data switch Network Administrator.

- **Host protocol error**

  The call could not be completed because of an error detected in the host protocol.

- **Insufficient CIR at module**

  The call could not be completed because the module receiving the call does not have sufficient resources to support the requested committed information rate (CIR).

- **Invalid or missing phone number**

  The user has neglected to type the phone number required to make a phone call as in **dkcu** *dialer_name.phone_number*. The *phone_number* is synonymous with the dialstring.

- **Mismatched GOS endpoints**

  The call can not be completed because the originating and receiving end-points of a call have the grade of service (GOS) configured differently.

- **Network congestion--Call forward error. Try again later**

  The call cannot be completed due to a lack of resources needed by a trunk in the call path. Try again.

- **Network congestion--Call initiation failure. Try again**

  The call cannot be completed because a message usually sent between processes during call setup was not sent. Wait a short time then call again.

- **Network congestion--Call timeout. Try again.**

  The call cannot be completed. No response was received and the call attempt timed out. Try again.

- **Network congestion--Channel allocation error. Try again later.**

  The call cannot be completed because there is no free channel in the data switch control computer database. Wait a short time before trying to call again.

- **Network congestion--Connection error. Try again later**

  The call cannot be completed because a connection cannot be made. Wait a short time before trying to call again.

- **Network congestion--Trunk call collision. Try again**

  The call cannot be completed because of a call collision in the trunk. Wait a short time and try the call again.

- **Network hung up**

  The network hung up the call.

- **Network routing error**

  Due to a network configuration error, the call is being sent and received over the same trunk.

- **No answer from dialed number**

  The call was dialed successfully but the autodialer did not get an answer.

- **No carrier tone detected**
  No carrier tone was detected

  The call was dialed successfully and answered; however, no carrier tone was detected by the autodialer.

- **No diagnostic channel**

  The call could not be completed because a diagnostic channel is not available.

- **No initial dial tone. Try again later.**
  No initial dial tone detected

  The autodialer on the called port may have a bad telephone line. The port has been marked as bad and removed from the autodialer hunt group.

- **No response from autodialer. Try again**

  The autodialer on the called port failed to respond to the data switch node. The port has been marked as bad and removed from the autodialer hunt group.

- **No secondary dial tone. Try again later.**
  No secondary dial tone detected

  The autodialer was signaled to wait for a secondary dial tone in the dialing sequence and no such dial tone was detected.

- **Open channel error**

  The call could not be completed because an error occurred opening the channel.

- **Please supply a valid phone number**

  The user has neglected to type the phone number required to make a phone call as in **dkcu** *dialer-name.phone-number*. The *phone-number* is synonymous with the dialstring.

- **Receive window too small**

  The call could not be completed because the receive window size is too small.

- **Remote node not answering**

  A connection cannot be completed because a data switch trunk or common data switch control computer database somewhere in the network path is inoperable.

- **Server already exists**

  This host attempted to set up a CommKit Host Interface server, but the data switch control computer database already knows a server by that name.

- **Server not answering**

  The requested server name is not in service. No interface hardware is assigned to the requested name. The interface hardware assigned to the requested name is not in service or is not operational. The remote server may not answer for reasons of its own. See *dkserver(1M)*.

- **Service mismatch**

  The service on the SIM did not match the request.

- **SIM all channels busy**

  All assigned ports/channels on the SIM are in use.

- **SIM bad contact**

  The call could not be completed due to SIM resource congestion.

- **SIM no access**

  The call was denied by the remote side.

- **SIM no contact**

  The SIM rejected the call.

- **Splice completed**

  The connection endpoints were successfully spliced together.

- **Splice failed**

  The connection endpoints could not be successfully spliced together.

- **Transmit window too big**

  The call could not be completed because the transmit window size is too big.

- **Trunk busy**

  The call could not be completed because a trunk in the call path is busy.

- **Trunk configuration mismatch. Call Network Administrator**

  The call was rejected because a trunk in the call path is not properly configured. Contact the data switch Network Administrator.

- **Trunk not answering**

  The call could not be completed because a trunk in the call path is not in service.

- **Unsupported baud rate**

  The requested baud rate is not supported by the autodialer.

- **Urp error**

  The call could not be completed because of an internal protocol error.

- **Vlp to dial error**

  The call could not be completed because an error occurred converting vlp format into a dialstring.

Printer Administration

This section describes the procedures for connecting, administering, and troubleshooting a printer on a data switch network. This section assumes that you have a working knowledge of the Line Printer (LP) Spooling Utilities. Refer to the *UNIX System V Release 4 System Administrator's Guide* for information on

administering LP Spooling Utilities.

UNIX System V Release 4 has printer set-up menus under the standard Operation, Administration, and Maintenance (OA&M) interface. Within the menus provided by OA&M there are menu selections to *Configure Printers on Remote Machine.* This selection controls other distributed printing features of UNIX System V Release 4 that work over TCP/IP and other supported protocols.

These OA&M programs eventually execute standard ***lp*** commands which are still supported in UNIX System V Release 4. This document will provide the traditional ***lp*** commands (***lpadmin, lpsched, accept,*** and ***enable***, etc.) so that if you are networking a group of UNIX System V Release 4 and earlier UNIX hosts, you can use one consistent set of ***lp*** commands on all the hosts. If you wish to use the OA&M to configure your printers, however, you may do so. Consult your UNIX System V Release 4 documentation for OA&M administration of printers.

The procedures that follow define set up procedures for CommKit Host Interface software running on UNIX System V Release 4. You can interconnect hosts of various releases together to share printers. The printer set-up procedures for earlier UNIX system releases will be slightly different. Refer to the documentation for those systems for specific information.

Sharing a Printer on a Data Switch Network

To allow several hosts to share a printer in the data switch network, one host must be configured as a spooling host and all other hosts on the network must be configured as remote hosts to eliminate printer contention. A spooling host is defined as the only host on the network that can receive print jobs from a remote host and schedule them on a printer connected to the data switch node or to the host itself. A remote host is a nonspooling host that uses the **dkdo** command to send its print jobs to the spooling host.

There are at least three ways to connect a printer to a data switch network. Each has advantages and disadvantages, and require different interface hardware and configuration procedures.

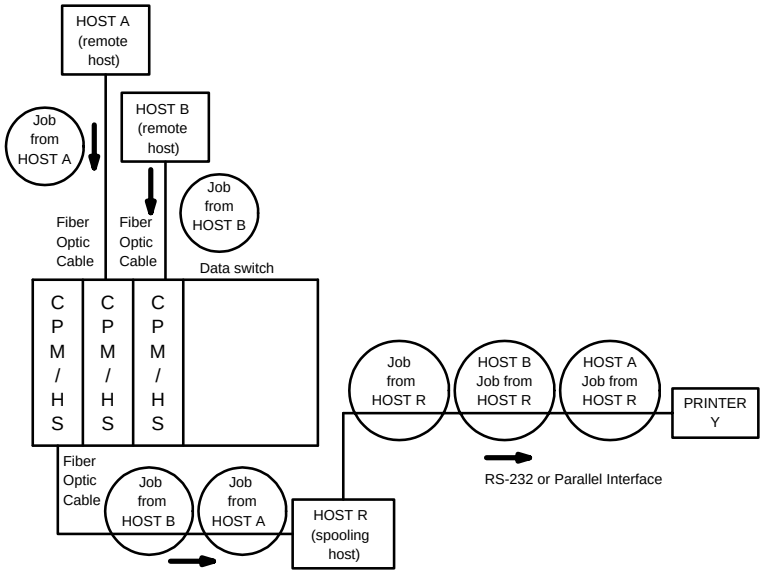
Before continuing with this section it may be helpful to read the *dkdo(1C),* *dkcat(1C),* and *dkdotab(4)* manual pages.

Printer Configurations

Configuration 1: Connection to a Local Host

Figure 4-3 shows the configuration of a printer directly connected to one host. This is the simplest way for hosts interconnected with a data switch to share a printer. No special configuration of the data switch or the UNIX ***lp*** software is needed. In this case all hosts use a printer directly connected to one host. That is, the printer is connected to one host's serial or parallel interface independent of the network. The host with the printer connection is designated as the spooling host.

Figure 4-3: Printer Directly Attached to One Host

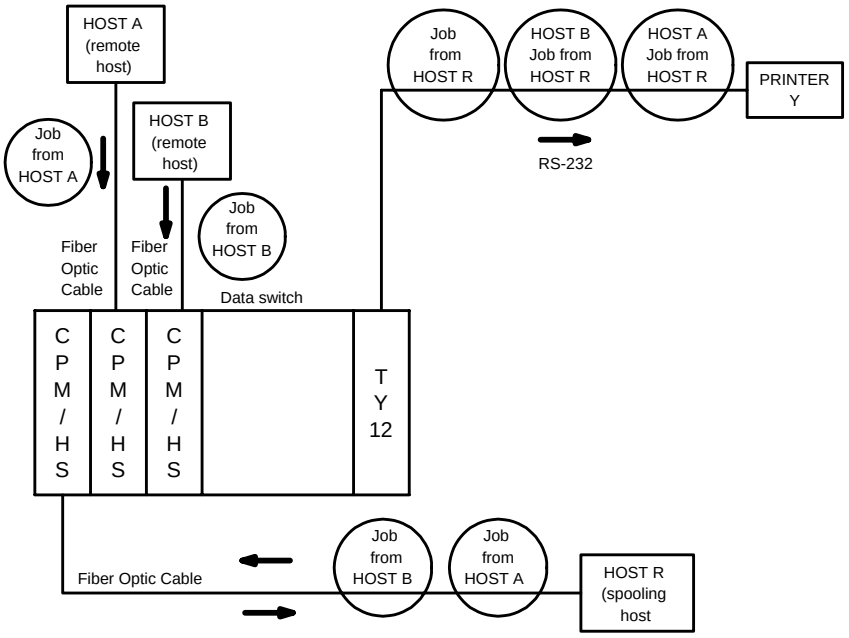


In this configuration, the remote hosts, **Host A** and **Host B**, send their print jobs to the spooling host, **Host R**, using the **dkdo** command. **Host R** spools the print jobs from the remote hosts and sends the print jobs to the printer (**Y**) using the ***lp*** service.

Configuration 2: Connection to a Data Switch Node; Spooling Host Using Fiber

Figure 4-4 shows the configuration of a printer connected to a spooling host which, in turn, is connected to the data switch network via the fiber interface. In this configuration, the spooling host uses **dkcat** to send the print jobs to the printer.

Figure 4-4: Network-Connected Printer; Spooling Host Using dkcat



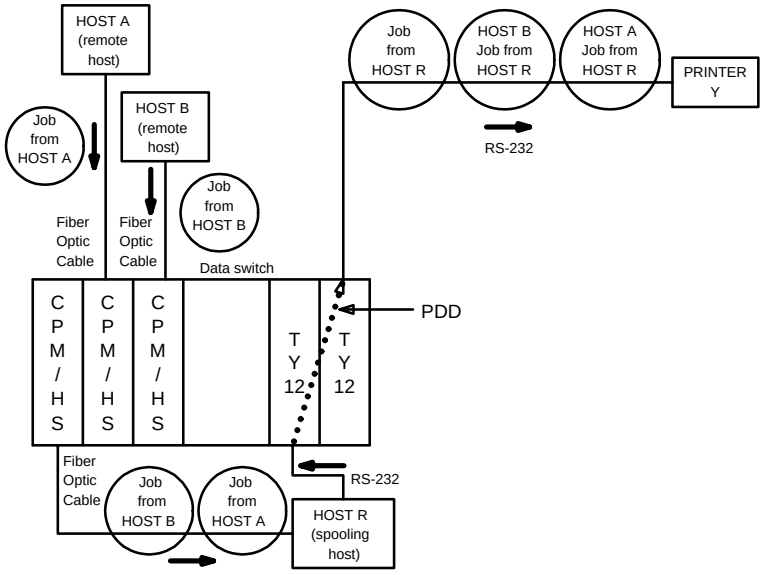
In this configuration, the remote hosts, **Host A** and **Host B**, send their print jobs to the spooling host, **Host R**, using the **dkdo** command. **Host R** spools the print jobs from the remote hosts and sends the print jobs to the printer (**Y**) using the **dkcat** command.

Note: In most cases, the ***lp*** subsystem is used to control the print jobs. An interface program (*model* file) must be used to have ***lp*** send the print jobs using **dkcat**.

Configuration 3: Connection to a Data Switch Node; PDD Connections

Figure 4-5 shows the configuration of a data switch network with the spooling host interfaced with the printer using the RS-232C port. With this configuration, the spooling host uses an asynchronous tty port and a Predefined Destination (PDD) to send a print job to the printer rather than **dkcat**. This configuration allows you to receive status messages back from the printer (such as a PostScript Printer). The RS-232C port on the spooling host is connected to the network by a TY12 port, and a PDD is set up from the spooling host TY12 port to the printer TY12 port.

Figure 4-5: Network-Connected Printer Using PDD



In this configuration, the remote hosts, **Host A** and **Host B**, send their print jobs

to the spooling host **Host R** using the **dkdo** command or other remote execution facility. **Host R** spools the print jobs of the remote hosts and sends them to the printer (**Y**) using the ***lp*** command over the asynchronous tty port and PDD.

Note: Although the spooling host has a direct virtual circuit to the printer by means of the PDD, the printer is also accessible using the **dkcat** command. That is, the printer can still be known to the data switch node through its service name, if set up to do so.

Configuration Procedures

Remote Host Configuration

To set up the remote host for all configurations:

1. Log on as **root**.
2. Save the original version of ***lp*** and ***lpstat*** by entering the following:

```
# mv /usr/bin/lp /usr/bin/Olp
# mv /usr/bin/lpstat /usr/bin/Olpstat
```

3. Link **dkdo** to ***lp*** and ***lpstat*** by entering the following:

```
# ln -s /opt/dk/bin/dkdo /usr/bin/lp
# ln -s /opt/dk/bin/dkdo /usr/bin/lpstat
```

4. If you wish to allow users to cancel ***lp*** jobs from the remote system, save the original version of */usr/bin/cancel* and link **dkdo** to ***cancel*** by entering:

```
# mv /usr/bin/cancel /usr/bin/Ocancel
# ln -s /opt/dk/bin/dkdo /usr/bin/cancel
```

5. Add the network address of the spooling host to the */etc/opt/dk/dkhosts* file.

```
system dflx dialstring -
```

system is the name of the spooling host.

dflx is the class of services required:

d — dkdo
f — file transfer
l — remote login
x — remote execution

dialstring is the network address of the spooling host.

– is a null field.

6. Define the ***lp*** service by adding the following to the */etc/opt/dk/dkdotab* file:

```
lp       system    -    d:n:o:t      -c:<*
lpstat   system    -       -           -
```

where *system* is the name of the spooling host as specified in the */etc/opt/dk/dkhost* file.

Note: Refer to *dkdotab(4)* manual page for a description of other fields.

7. For the ability to cancel an ***lp*** job from the remote host, add the following to */etc/opt/dk/dkdotab*:

```
cancel   system   -     -     -
```

Spooling Host Configuration Procedures

Local Host — Configuration 1

To set up the spooling host for the configuration shown in Figure 4-3 follow the steps below.

1. For the spooling host to receive jobs from the remote host using the **dkdo** command, the */etc/opt/dk/srvtab/do* file must contain:

```
*  do   R/vx   *n>9   /opt/dk/bin/dkdo   dkdo:-p
```

where ***** allows all hosts from all areas and exchanges to make **dkdo** requests. You may wish to limit which machines have such permissions. See *srvtab(4)* manual page.

2. In the previous step, it is assumed that all users on the remote hosts have logins on the spooling host, and that their user IDs match. If you wish to allow users from remote hosts who do not have a login on the spooling host to print jobs, create a **guest** user login on the spooling host and add an entry to the */etc/opt/dk/srvtab/do* file with the following:

```
*   do   R/vx   guest    /opt/dk/bin/dkdo    dkdo:-p
```


Data Switch Node (Spooling Using Fiber) — Configuration 2

Set up the **do** service by completing steps 1 and 2 under *Local Host — Configuration 1*. Refer to Figure 4-4. The spooling host must be set up to use **dkcat** to send jobs to the printer. To configure the standard UNIX system ***lp*** code, the user must be logged on to the UNIX system as **root** or ***lp***. For the ***lp*** system to access the printer on a data switch, a number of UNIX system files must be edited or

created. Some of these files provide printer-specific information and are used by the ***lp*** system; others are used by the CommKit Host Interface software to access the printer on the data switch network. The files are described in the paragraphs below.

The *model* file is a shell file (see *sh(1M)* in the *UNIX System V System Administrator's Reference Manual*) used to interface the line printer spooling system to a specific type of printer. This file is found in the directory */etc/lp/model*. The file is created based on the specific characteristics of the printer, so that all control characters are properly interpreted. The *model* file is designed to be shared by printers of the same type; when an additional printer is added, the *model* file is copied into the *interface* file with the same name as the name given to the printer.

The *model* file must be modified to invoke the **/opt/dk/bin/dkcat** command. Either the output of the printer-specific script may be piped into the **dkcat** command or the file name may be provided as an argument to **dkcat**. The **dkcat** command sends files from the host to a printer connected to a network endpoint. The *model* script must exit with the exit code returned by **dkcat**. Refer to the *dkcat(1C)* manual page for more information.

Create a *model* file and place it in */etc/lp/model/printer_name*. The **printer_name** is the name you wish to use to access the printer. A sample model file for a dumb serial printer is shown in Figure 4-6.

Figure 4-6: Sample model File

```
#  lp Model file for a dumb serial line printer connected over
#  data switch.  The name of this model file should be the node
#  service name for the printer. The script is enclosed in ( ) and
#  its output is piped into the dkcat command. The lp subsystem
#  executes this script with arguments as follows:
#
#  arg1     - Print job number
#  arg2     - The machine/login in the format host!user
#  arg3     - Additional banner information from lp -t
#  arg4     - Number of copies
```

Figure 4-6. *Continued*

```
#  arg5     - Not used
#  arg6-rN   - full pathname  of file(s) to be printed
#
# Some lines contain samples of how to save debug information in a file
# like /tmp/printlog. To get debugging turn on lines such as the following:
#
# echo `date` job "$1" user "$2" >> /tmp/printlog

printer=`basename $0`
x="xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx"

(
echo "\014\c"
echo "$x\n$x\n$x\n$x\n"
banner "$2"
echo "\n"
user=`grep "^$2:" /etc/passwd | line | cut -d: -f5`
if [ -n "$user" ]
then
        echo "User: $user\n"
else
        echo "\n"
fi
echo "Request id: $1    Printer: `basename $0`\n"
date
echo "\n"
if [ -n "$3" ]
then
        banner $3
fi
copies=$4
echo "\014\c"
shift; shift; shift; shift; shift
files="$*"
i=1
while [ $i -le $copies ]
do
        for file in $files
        do
                cat "$file" 2>&1
                echo "\014\c"
        done
        i=`expr $i + 1`
done
echo "$x\n$x\n$x\n$x\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n"
```

Figure 4-6. *Continued*

```
echo "\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n$x\n$x\n$x\n$x"

# uncomment sleep 15 if last lines of output
# are lost, in particular for slow devices or SAM endpoints
# sleep 15
# Change /dev/null to /tmp/printlog to debug problems on line below
) | /opt/dk/bin/dkcat $printer 2>&1 >>/dev/null
retcode=$?

# next line shows how the exit code of dkcat can be logged
# echo "Job $1 returned $retcode\n" >>/tmp/printlog

# the return code $? from dkcat is returned to the lp system.
exit $retcode
```

The *interface* file is needed for each printer known to ***lp***. This file is a shell file located in the */etc/lp/interfaces* directory. It is a duplicate copy of the *model* shell file and is executed by ***lpsched*** to send a job to the printer. This file will be created by the ***lpadmin*** command later using the *model* file as input.

The *dkhosts* file allows the CommKit Host Interface software on the spooling host to access the printer on the network. The printer name and its network dialstring with class **p** must be entered in the file */etc/opt/dk/dkhosts*. For example, for the **demoprt** printer, use an entry such as the following:

```
demoprt     p     nj/localnet/demoprt      -
```

For more information, refer to the *dkhosts(4)* manual page. This information is used by the **dkcat** command to establish a connection to the printer. The printer must be given a unique destination **name**. The **name** must be the same as the **Mnemonic Address/Service Name** given to the TY/SAM endpoint where the printer is connected. The printer can be configured on the spooling host by entering the following commands on the spooling host:

```
# /usr/lib/lpshut
# /usr/lib/lpadmin  -pprinter_name  -mprinter_name   -v/dev/null
# /usr/lib/lpsched
# /usr/lib/accept  printer_name
# enable printer_name
```

- *printer_name* is the name of both the *model* and *interface* files used to interface the line spooling system to a specific type of printer.
- **-m** option is used if the *model* file exists in the */etc/lp/model* directory.
- **lpadmin** command copies the */etc/lp/model/printer_name* file to the *printer_name* file in the interface directory */etc/lp/interfaces*.
- **-v** option specifies the special character device where the printer is connected. Since the printer is being accessed by **dkcat**, this device is always the null device (**/dev/null**).

To send a print job to the printer, ***lpsched*** opens the printer device (specified by the **-v** option) and makes it **stdin** and **stdout** and executes the *interface* file for the printer. Refer to the *UNIX System V Release 4 System Administrator's Guide* for more details on the ***lpadmin*** command.

dkcat Notes

The utility program, **dkcat**, is designed to enable the transmission of files to remote printers connected to the data switch network. The **dkcat** command is invoked in the *model* file with the data switch **destination** of the remote printer and other options, which can include the file to be printed. The specified **destination** is mapped into a full dialstring by means of the */etc/opt/dk/dkhosts* file.

Certain printers, such as Imagen Laser Printers, require character translations. The **-I** option provides that translation in the **dkcat** program. Every character in the file is examined, and if a character is found to be one of two special characters – 0x9e (**eof** character) or 0x9b (**quote** character) – special processing occurs.

To override the definition of the **eof** character and / or the **quote** character, use the **-q** option. **dkcat** takes an argument of the form <**quotechar**>,<**eofchar**>. There must be no spaces between the character definitions and the comma; the character definitions must be in decimal format. The decimal values for 0x9e and 0x9b are 158 and 155, respectively.

If the **-F** option is specified, a form feed character (0xc) is sent after each file. This option is used to make the beginning of each file start on a new page.

Additionally, **dkcat** does *not* provide any traditional **stty** function. Much of the **stty** function is meaningless (e.g., baud rates and parity settings). However, other capabilities such as conversion of CR characters to CR and NL characters or expansion of tab characters into spaces may be needed. These can be easily performed in the *model* file by using a standard UNIX system command such as **sed** as a filter for the data to be sent to the printer. Alternately, most printers can be configured to do such translation either by jumper or switch settings or by sending them special codes. Refer to the printer manual.

dkcat does not read data from the data switch network and, hence, does not support any flow control mechanisms. Flow control mechanisms are provided between the TY/SAM endpoints and the connected printer. This means, in particular, that printers which write status messages back (such as PostScript Printers) are not well supported by **dkcat**. To support such printers, users should connect the printer directly to the spooling host as shown in Figure 4-3 or use PDD as shown in Figure 4-5.

dkcat exits with a non-zero value if it fails to establish a connection to the printer or if it encounters an error in data transmission. Refer to the section *CommKit Host Interface Exit Codes* for the exit codes. The exit codes for command failures and data transmission failures can also be found in */usr/include/dkit/sysexits.h* on any host with CommKit Host Interface installed.

Data Switch Node (PDD Connected) — Configuration 3

Set up the **do** service by completing steps 1 and 2 under *Local Host — Configuration 1*. Refer to Figure 4-5. In this configuration the ***lp*** software sees the printer as directly connected to the spooling host serial port, therefore, no special **lp** administration is needed. There may be some flow control issues, however, refer to the section, *Printer Flow Control*, later in this chapter.

Data Switch Configuration Procedures

Connection to a Local Host

Refer to Figure 4-3. Because the printer is not connected to the data switch node, and the hosts on the network use standard **dkdo** services, no special setup is needed on the data switch node.

Connection to a Data Switch Node

Refer to Figures 4-4 and 4-5. To configure the printer on the data switch node follow the steps below.

1. Execute the ***remove ty, change ty*** and ***restore ty*** commands from the data switch console. The system will prompt for additional information. The following is an example of configuring a printer with the ***change ty*** command:

Note: BAUD RATE, PARITY, and FLOW CONTROL should be set to values needed for your specific configuration.

```
MODULE ADDRESS:26
CHANNEL NUMBER:4
COMMENT:"up to 60 chars double quoted"
SERVICE TYPE:host
GROUP:lpprint
BAUD RATE:9600
EXTERNAL BAUD RATE:2400
PARITY:off
FLOW CONTROL OF TY-12 BY DEVICE:xon_xoff
FLOW CONTROL OF DEVICE BY TY-12:xon_xoff
IS AN AT&T VDM CONNECTED TO THIS LINE:no
```

2. The following is an example of configuring the PDD (refer to Figure 4-5) with the ***change ty*** command:

Note: BAUD RATE, PARITY, and FLOW CONTROL should be set to values needed for your specific configuration.

```
MODULE ADDRESS:26
CHANNEL NUMBER:5
COMMENT:"up to 60 chars double quoted"
SERVICE TYPE:terminal
GROUP:lporg
BAUD RATE:9600
EXTERNAL BAUD RATE:2400
PARITY:off
FLOW CONTROL OF TY-12 BY DEVICE:xon_xoff
FLOW CONTROL OF DEVICE BY TY-12:xon_xoff
IS AN AT&T VDM CONNECTED TO THIS LINE:no
NODE ECHOES USER INPUT:yes
CALL HOLD:off
PREDEFINED DESTINATION:demoprt
CONNECT-TIME BILLING:off
ATTENTION CHARACTER:none
```

where **demoprt** is the service name of the printer. The value of the PREDEFINED DESTINATION string should be the same as the printer address. The example given assumes that the group *lpprint* has been entered and that the printer address **demoprt** has also been entered and associated with the group *lpprint*.

Troubleshooting Printer Problems

The configuration chosen to connect the printer will dictate the troubleshooting approach. Table 4-4 lists the probable problems that apply to specific printer configurations.

Table 4-4: Troubleshooting Printer Problems

Configuration Printer connected to	Figure	Possible Problems
Local Host	4-3	• Printer problems • ***lp*** subsystem problems • **dkdo** problems.

Table 4-4: continued on next page

Table 4-4: *Continued*

Data Switch Node (Spooling Using Fiber)	4-4	• Printer problems • **dkcat** problems • ***lp*** subsystem problems, especially with the way **dkcat** is used in the *model* file • Configuration problems with the way the printer is attached to the data switch network • **dkdo** problems.
Data Switch Node (PDD)	4-5	• Printer problems • ***lp*** subsystem problems • Configuration problems with the way the printer is attached to the data switch network • Configuration problems when the spooling host's serial port is attached to the data switch network • Configuration problems with the PDD • **dkdo** problems.

Printer Problems

The printer configurations shown in Figure 4-3 (connected to a local host) and Figure 4-5 (connected to a data switch node using PDD) use the printer as though no network were involved. Refer to the UNIX System V Release 4 printer documentation to troubleshoot printers.

If the printer is connected to the spooling host serial port by a PDD:

- Disconnect the printer from the data switch
- Directly connect the printer to the host serial port
- Retest the printer functions by queuing an **lp** job on the spooling host.

If the printer does not work in this configuration, the problems are not related to the data switch node or CommKit Host Interface software.

dkdo Problems

The **dkdo** control files and commands are not configured properly if you are able to queue an **lp** job on the spooling host but are unable to send a print job from a remote host to the spooling host. If this occurs:

- Verify the general connectivity of the remote host to the spooling host. That is, can you perform **dk**, **dkcu**, and other CommKit Host Interface software operations? If not, check the data switch node configuration and the proper installation of the CommKit Host Interface software.
- Check the configuration of **dkdotab, srvtab**, and **dkhosts** entries.
- Verify that the ***lp*** and ***lpstat*** commands on the remote host have been linked to **dkdo**.

A simple test of the **dkdo** function can be done by adding an entry for a basic UNIX system command, such as ***date*** or ***uname*** to the *dkdotab* file on the remote host. The *system* field for the *dkdotab* entry should be the hostname of the local machine. Refer to the section *dkdotab* in Chapter 3 for more information. For example, add the ***uname*** command to the remote system's **dkdotab**, and execute the following on the remote system:

```
# dkdo uname -a
```

This will execute the ***uname*** command on the spooling host and the name and UNIX system configuration information from the spooling host should be printed on your terminal. If this operation fails, then your remote host to spooling host connections need to be investigated.

If jobs correctly queue for some users, but not others, it may be that user logins on the remote host are unknown on the spooling host. That is, the **srvtab** entries are such that the remote execution of ***lp*** by **dkdo** (that is, the **do** service) is being refused. If users on the remote systems do not have logins in the spooling host, create a **guest** login on the spooling host and set up **srvtab** to map unknown user logins to that login.

dkcat Problems

When using the printer configuration shown in Figure 4-4 (connected to a data switch node using fiber), a printer *model* file must be created on the spooling host that uses **dkcat** to send a print file to a printer connected to the network. This arrangement requires the ***lp*** control programs (***lpsched***, etc.) to be properly configured and running. If ***lp*** jobs are not being sent to the printer, the first step is to see if the print jobs are being properly queued. The ***lpstat -t*** will display the status of printers and jobs. If print jobs are being properly queued but not being sent to the printer, try connecting to the printer using **dkcat** manually. That is, execute the following in the spooling host:

```
# banner this is a test | dkcat printer_name
```

where *printer_name* is the service name of your network-connected printer. If the banner *this is a test* appears on the printer (without the usual ***lp*** header) this means that the data switch node configuration, **dkcat** command, and CommKit Host Interface software are all operating correctly and it is the ***lp*** software or configuration that is at fault. It is important to check the *model* file. Try manually executing the lines in your *model* file that call **dkcat** and see if any errors are displayed.

Before doing an exhaustive test of **dkcat**:

- Verify that the printer is properly configured.
- Verify that the printer is on-line with no paper or ribbon faults.
- Execute the printer's local test mode, if it has one.
- Enter the service name of your printer on any terminal connected to the data switch. After a connection has been made to the printer additional characters typed on the terminal should appear on the printer.

Returns an Error

The exit code of **dkcat** gives a reason for failure; refer to the section *CommKit Host Interface Exit Codes.*

Fails to Make a Connection With the Printer

The **stderr** message usually gives enough information to investigate the cause of a connection failure. If the printer is found to be busy, make sure there is no contention for the printer port from multiple sources in the network. Two or more hosts may be establishing a connection to the same printer. Examine the appropriate data switch configurations to determine what hosts are directly interfacing to the printer. Make sure only one host is configured as the spooling host for the printer.

Data Transmission Problem

There are two common reasons for this problem: the spooling host may be running out of resources or a connection is broken in the network during data transmission. Using **dkcat**, send a file to a terminal which is known to be functioning properly. If **dkcat** still fails during data transmission, it is most likely that the system is running low on resources. Check the value of the system tunable parameters. Use the **strstat** function of the ***crash*** command to look for problems with STREAMS resources.

If **dkcat** succeeds (exit code 0) but data is still being lost, it may be the printer connection which is incorrect. In this case, monitor the network port to printer line using a breakout box or any other appropriate equipment and check if DTR is being toggled while a job is being transmitted. Check for any other intermittent signals.

Works but Partial Data Loss

If printer output indicates that data is lost in different sections of the file, the flow control setting must be checked between the network port and the printer. If flow control is working correctly, recheck the hardware (i.e., network port, printer cable, printer connector, and printer itself).

If printer output indicates that only the end section of the file is lost, check the printer and monitor the line between the network endpoint and the printer to see if the remaining section has been sent to the printer and is in the printer buffer. If the printer is found off-line and putting it on-line prints the remainder of the file, the problem may be the printer connector.

Printer Flow Control

As a flow-control mechanism, some printers send XON/XOFF characters to the network and also toggle DTR. Since dropping DTR causes the network endpoint to drop the network connection, a connector change is required for these printers so that DTR remains high during flow control. To do this, tie RTS (pin 4) to DTR (pin 20) from the printer and sever the original DTR connection from the printer.

lp Subsystem Problems

If the complete job prints correctly when sent from **dkcat** directly, but not when the ***lp*** command is executed on the spooling host, the problem may be in the ***lp*** subsystem. Recheck the *model* file and check the correct operation of ***lpsched, lpadmin, accept, enable***, and the rest of the UNIX system ***lp*** commands. Refer to the UNIX System V Release 4 documentation for additional ***lp*** information.

CommKit Host Interface Exit Codes

The following table lists the CommKit Host Interface command exit codes. When a command such as **dkcat** fails to execute properly, the ***sh*** will receive a non-zero exit code which you can display by entering ***echo $?***.

Return Code	Meaning
0	successful termination
2-64	command syntax errors
65	data format error
66	cannot open input
67	addressee unknown
68	host name unknown
69	service unavailable
70	internal software error

71	system error (e.g., can't fork)
72	critical OS file missing
73	can't create (user) output file
74	input/output error
75	temp failure; user is invited to retry
76	remote error in protocol
77	permission denied
78	all channels busy
79	remote node not answering
80	server not answering
81	all trunk channels busy

5 Compatibility

Introduction

This chapter is included as a reference for programmers who may be porting applications from a UNIX System V Release 3 environment; it also contains useful information and examples for programmers who are developing new applications using the CommKit Host Interface.

This chapter discusses the compatibility issues associated with the current CommKit Host Interface software release operating with UNIX System V Release 4. The compatibility issues are divided into three sections; environment variables, user-level, and programmer-level. The environment variables section describes the shell variables supported by the host interface and their effect on user commands and programmer interfaces. The user-level section addresses the relationship of the user-level commands to the STREAMS architecture of the underlying software. The programmer-level section addresses the new library and system call interfaces as well as how to port existing application programs to the new architecture. This new interface has been designed to hide implementation details and to minimize future porting efforts.

The overall function of the CommKit Host Interface software for UNIX SVR4 is identical to that of CommKit Host Interface software releases for UNIX SVR3. Applications that use the user-level commands may have to account for the new UNIX system features but may continue to interface with host commands as they have in previous releases. Applications that use the library and system call interfaces will need to be reprogrammed. The reprogramming effort will, however, yield cleaner and more maintainable code.

Note: RFS is not supported by the NCR UNIX System.

Environment Variables Compatibility

All previous CommKit Host Interface software releases have supported the environment variables DKINTF and DKKEY. The variable, DKGROUP is introduced in the current release. Table 5-1 summarizes the status of these variables for the current release.

Table 5-1: Status of Environment Variables

Variable	Status
DKKEY	unchanged
DKINTF	definition changed
DKGROUP	new

Note: If your previous CommKit Host Interface release supported the environment variable DKNUMINTF, this function is included in DKINTF. To support the old function of DKINTF and DKNUMINTF with two interfaces, DKINTF can have the values of **0,1** or **1,0**. (See the section *DKINTF* below.)

DKINTF

DKINTF controls the interface used to place outbound calls by means of **dkdial**. Previously, if DKINTF was not set, the outbound call would be placed on interface 0.

All SVR4 releases perform round-robin dialing over all active interfaces. This feature applies to all dial attempts in the system and is not per user.

Note: For applications requiring pre-SVR4 behavior, set DKINTF to 0 and export it.

This CommKit Host Interface release supports a new syntax for DKINTF (a comma-separated list of interfaces) as shown below:

```
DKINTF=0,0,1
```

The dial-out attempts are made in the order defined in the list until a call is completed or until the end of the list is reached. This allows lists in which one interface is preferred over others. For example, the list shown in the screen fragment above will dial over interface 0 twice before dialing over interface 1.

DKGROUP

DKGROUP controls the channel groups used to make a connection to a remote endpoint. The use of channel groups allows administrators to configure multiple groups for a single CPM-HS module which allows greater flexibility in originating group screening on remote systems. Channel groups do *not* require different groups on the AT&T data switch. When used this way, channel groups can control how many connections can be made using the group. The definition of channel groups does not restrict access to the channel group based on UNIX filesystem permissions, however, the administrator can adjust the *dkitrc* script to specify permissions.

The administrator defines channel groups in the control file *dkgroups* which defines the name, interface(s), and channel range for the group. Users can then make connections using a channel group definition by setting and exporting the DKGROUP variable to an entry in the *dkgroups* file.

Refer to the *dkgroups*(4) manual page for more details.

The system administrator can set the DKGROUP variable during login processing in */etc/profile* on a per-user basis. By making DKGROUP read-only, it will restrict what channels and interfaces are available to a user.

User-Level Compatibility

The CommKit Host Interface user-level commands contain the same functions as previous releases. The major compatibility issues are the expanded features of the UNIX System V Release 4 and their backward compatibility to UNIX System V Release 3.2. Other compatibility issues are related to the new STREAMS architecture of the current CommKit Host Interface software release. The features that affect the CommKit Host Interface commands are:

- Symbolic links
- Job control
- Long file names
- STREAMS

The following sections describe the relationship of these features to the commands.

dk

The **dk** command implements remote execution and remote login between the local and the remote host. The operation of the **dk** command can be affected if the local and remote host are running different releases of the UNIX system, because the **dk** command is designed to support only processes that conform to the base UNIX System V definition. Processes that do not conform to this definition, even though they are supported by UNIX System V Release 4, may behave poorly under a remote execution or remote login session.

The **dk** command is affected by different releases of the UNIX system as shown below:

- The remote execution and remote login commands do not use a STREAMS interface on the remote host. Commands that depend on a STREAMS interface may not work.
- Commands given as the arguments with the **dk** command cannot be controlled by the UNIX System V Release 4 job control mechanism. See *sh(1)* in the *UNIX System V Release 4 User's Reference Manual* for more information on job control. The **dk** command running on a local host with UNIX System V Release 4 may be controlled by the job control mechanism allowing the user to execute multiple **dk** commands. Each execution of the **dk** command uses one data switch connection.

When the **dk** command is run by a shell that supports job control [e.g., *ksh*(1) or *jsh*(1)], the default behavior is to support job control. When **dk** is run in the background, all *stty*'s that can set the controlling terminal and all data written to the terminal will be blocked until the **dk** command is moved to the foreground. This is generally desired in background programs because this behavior does not cause controlling terminal problems. In cases where this blocking is *not* desired, use the **-j** option to prevent blocking on *stty*'s and *write*'s. Using the **-j** option is the same as using */bin/sh* and running **dk** in the background.

dkcat

The **dkcat** command is fully compatible with earlier releases of CommKit Host Interface and the UNIX system. Current applications that use **dkcat** need not change their interface with the **dkcat** command.

dkcu

The **dkcu** command is fully compatible with earlier releases of the CommKit Host Interface and the UNIX system. Current applications that use **dkcu** need not change their interface with the **dkcu** command.

Note: Long file names are not given special treatment when using the **%put** and **%take** features of the **dkcu** command. If a source file has a name longer than 14 characters and the target system is running UNIX System V Release 3.2, the file name will be truncated.

As an added security feature, a restricted option (**-r**) is available to disable the **~** escape sequences (with the exception of **~.** and **~break**) for **dkcu**. This option is useful in controlling user access to systems by restricting the escape sequence to disconnect or break only.

dkdo

The **dkdo** command is fully compatible with earlier releases of the CommKit Host Interface and the UNIX system. Current applications that use **dkdo** need not change their interface with the **dkdo** command.

push and pull

The **push** and **pull** commands are fully compatible with earlier releases of the CommKit Host Interface and the UNIX system but they have been enhanced to support the transfer of symbolic links and long file names. Current applications that use **push** and **pull** need not change their interface with these commands.

push and **pull** preserve the name, type, and contents of the files they move with the following exceptions:

Long Names — If both the source and destination systems are running UNIX System V Release 4 and the destination file system is of type S5, and the source system sends a file name greater than 14 characters, then the destination system will truncate the file name to 14 characters and send a warning message notifying the user that the file name has been truncated.

If the source system is running UNIX System V Release 4 and the destination system is running UNIX System V Release 3.2, and the source system sends a file name greater than 14 characters, then the file name will be truncated to 14 characters and the source system will send a warning message notifying the user that the file name has been truncated.

Symbolic Links — Symbolic links will be preserved when the destination supports them. If the destination machine does not support symbolic links, then the source machine will take the following actions:

- If the symbolic link points to an ordinary file, the transferred file will be an ordinary file with the name of the symbolic link but with the contents of the ordinary file.
- If the symbolic link points to a directory, the transferred file will be a directory and the source system will treat it as a directory and transfer all of the files it finds in that directory on the source system.

  Because symbolic links that point to directories may be circular paths, the source system limits the extent to which it will follow a directory path that contains a symbolic link. In particular, if the path contains a symbolic link and either the directory pointed to that link, or one of the sub-directories has already been transferred, the source system will not transfer it again. The source system will advise the user of the status of every symbolic link that points to a directory.

Programmer-Level Compatibility

The architectural changes of this release will impact all applications that access the CommKit Host Interface software through the library and system call interface.

In particular, there is no **ioctl** interface supported in the current CommKit Host Interface software release. All **ioctl** calls are reserved for use by the interface library and their calling sequences are not guaranteed.

Porting old applications to the new release will, however, yield simpler, more robust code. The sections that follow discuss the changes and show examples of the new library routines and system calls. Many of the examples are taken from the CommKit Host Interface user-level commands.

STREAMS

Application writers should be familiar with the characteristics of STREAMS I/O and the *UNIX System V Release 4 STREAMS Programmer's Guide.* In general, applications that use the standard I/O library function calls (except to account for STREAMS errors) do not need to be changed.

One aspect of STREAMS that application writers should be aware of is the hand-off nature and internal buffering of writing data. A **write** system call to a Stream will return immediately from the Stream head and the data will be queued to the downstream modules and drivers. If the network is very busy or the driver cannot transmit the data, the Stream will eventually flow control, but only after multiple **write** system calls complete. Refer to the section *write(2)* later in this chapter for more information.

TTY Interface

The current CommKit Host Interface software release, unlike previous releases does not reimplement the UNIX system tty interface. Instead, it uses the standard TTY STREAMS modules, **ldterm** and **ttcompat**.

Note: The tty devices in the */dev/dkt* directory are used for accounting purposes only and only provide a tty interface for incoming calls accepted by the **dkserver** process.

Message Boundaries

The current CommKit Host Interface software release preserves message boundaries as they are received from the network. This means that all incoming URP GOS1 and GOS2 data is available immediately and all URP GOS5 (that is, block mode) data is available after the appropriate verification and processing. The block mode message boundaries are preserved with the following exceptions:

- Messages are split at every level-D URP code. (Refer to *read(2)* in the *System Call Compatibility* section later in this chapter.)
- Messages may not be larger than the **dkhs** tunable configuration parameter *dkhs_rcv_max_msg_sz* (default 10240). Messages received that are larger than this parameter are split into two or more smaller messages at boundaries convenient to the **dkhs** driver.
- The kernel splits transmitted messages so that their length does not exceed STRMSGSZ bytes. For example, a 6144-byte write will be split into two messages – 4096 bytes and 2048 bytes – when STRMSGSZ is 4096 bytes. This parameter is found in the */etc/master.d/kernel* file. Set STRMSGSZ to the same value as *dkhs_rcv_max_msg_sz* to make the maximum transmitted and received message sizes more even.

  Applications that rely on preserving specific message boundaries (e.g., **dkload**) may suffer adverse affects from setting the value of STRMSGSZ less than *dkhs_rcv_max_msg_sz*.

Header Files

This section describes the header files that may be needed when writing application level programs to operate with the CommKit Host Interface. The support for the contents of these files is limited to the **#define** entries and the structures and global variables listed on the library manual pages. The following is a description of the header files you may need:

dk.h This header file contains the error codes, external declarations, and default file names used by the current CommKit Host Interface software release. All applications that use the CommKit Host Interface to place calls should include this header file.

dk_urp.h This header file contains the URP level-D codes used by the **dkleveld** library function.

rdfp.h This header file contains the **dk_lvld_t** definition used by the **dkleveld** library function.

sysexits.h This header file contains exit codes used by the current CommKit Host Interface software release user-level commands.

Library Interface Compatibility

Most of the old library routines have been replaced or dropped from the current CommKit Host Interface software release. The new library routines support the old actions and additional ones so that programmers will not have to wrestle with the use of the **ioctl** interface to the CommKit Host Interface drivers.

Obsolete Library Routines

The following routines are not supported for the current CommKit Host Interface software release:

dkminor This function converted a file descriptor into a data switch channel number. This function has been replaced by the **dk_info** function that returns both the interface and channel number. Refer to the *dk_info* example later in this chapter.

Note: This function was often used with the **dtnamer** function when switching an outgoing call from the raw interface to the tty interface. Converting the SVR3 old code to use the SVR4 equivalents will not make an outgoing call over the tty interface. A process obtains a tty interface directly by using the new **dkitdial** library function. Refer to the *dkitdial* example later in this chapter.

dknamer This function converted a CommKit Host Interface minor number into the **/dev** name for the raw interface. It has been replaced by the **dk_namer** library function that accepts both the CommKit Host Interface number and the data switch channel number as arguments. Refer to the *dk_namer* example later in this chapter.

dtnamer This function converted a CommKit Host Interface minor number into the **/dev** name for the tty interface. This function has been replaced by the **dk_tnamer** library function that accepts both the CommKit Host Interface number and the data switch channel number as arguments. Refer to the *dk_tnamer* example later in this chapter.

Note: This function was often used with the **dkminor** function when switching an outgoing call from the raw interface to the tty interface. Converting the SVR3 old code to use the SVR4 equivalents will not make an outgoing call over the tty interface. A process obtains a tty interface directly by using the new **dkitdial** library function. Refer to the *dkitdial* example later in this chapter.

dxnamer This function converted a CommKit Host Interface minor number into the **/dev** name for the remote execution interface. This function has been replaced by the **dk_xnamer** library function that accepts both the CommKit Host Interface number and the data switch channel number as arguments. Refer to the *dk_xnamer* example later in this chapter.

dkxwrite This function transmitted data to the remote system and was used by the remote execution interface. This function is no longer supported for user-level application programs.

Supported Library Routines

This section describes the library routines that are supported by the current CommKit Host Interface software release. These are the only supported interfaces available to application developers. All of the **ioctl** calls supported in previous releases are no longer supported.

These library functions should satisfy all of the CommKit Host Interface requirements of user-developed application code. Many of the new functions have been designed to isolate application code from the underlying driver and make it more portable to future releases of the CommKit Host Interface software. The library is delivered as a dynamically linked shared object library so that application processes will not have to re-compile or re-link when a new version of the CommKit Host Interface software is installed.

Because the CommKit Host Interface library is now a dynamically linked shared library, its name has been changed from */usr/lib/libdk.a* to */usr/lib/libdk.so*. The method of access, however, remains the same (that is, including **-ldk** on the compilation command line.) Table 5-2 describes the new library routines. In addition to providing a dynamically linked shared library, a static version (*/opt/dk/lib/libdk.a*) of the CommKit Host Interface library is available. This library can only be accessed by specifying the full path name of the library on the link edit command line. The static library *libdk.a* can be used to create static images of CommKit Host Interface applications, however, this static binding requires user applications to be link edited again whenever a new release or patch of the CommKit Host Interface library is installed. CommKit Host Interface library updates require a new application release to distribute the update, whereas the recommended dynamic binding shared library (*libdk.so*) does not require a new application release and does not need to be link edited again. For these reasons, the use of *libdk.so* (shared) is recommended over *libdk.a* (static).

Table 5-2: Supported Library Routines

Man Page	Function	Description
dk_flush(3X)	dk_flush()	Modifies the handling of user data queued for transmission when the CommKit Host Interface connection is closed. The user can now wait for the default time interval, specify a time interval, or wait until all data has been received by the URP receiver at the remote end of the CommKit Host Interface connection.
dk_info(3X)	dk_info()	Retrieves or sets information about an open CommKit Host Interface connection. At present the information returned is limited to the CommKit Host Interface number, the data switch channel number, the URP window size, and the number of URP blocks outstanding. The information that may be set is limited to the URP window size and the number of outstanding URP blocks.
dk_namer(3X)	dk_namer()	Converts a CommKit Host Interface number and a data switch channel number into the device name under */dev/dk* for the raw interface.
dk_namer(3X)	dk_tnamer()	Converts a CommKit Host Interface number and a data switch channel number into the device name under */dev/dkt* for the tty interface.

Table 5-2: continued on next page

Table 5-2: *Continued*

Man Page	Function	Description
dk_namer(3X)	dk_xnamer()	Converts a CommKit Host Interface number and a data switch channel number into the device name under */dev/dkx* for the remote execution interface.
dk_uxinfo(3X)	dk_uxinfo()	Retrieves or sets information about URP parameters of receive buffer size and number of outstanding urp blocks. This can only be used when the DKUX module is pushed on the STREAM.
dkdial(3X)	dkitdial()	May be used to place all outgoing calls. Its options include the selection of CommKit Host Interface, raw or tty device, and baud rate.
dkdial(3X)	dkdial()	Provides the traditional **dkdial** function.
dkdial(3X)	dkndial()	Provides the traditional **dkndial** function.
dkdial(3X)	dktdial()	Provides the traditional dial function on a tty device by calling **dkitdial**.
dkdial(3X)	dkntdial()	Provides the traditional dial function on a tty device for the given interface by calling **dkitdial**.
dkdial(3X)	dkgdial()	Provides outgoing calls on a tty device using a specified group.
dkdial(3X)	dkgtdial()	Provides outgoing calls on a tty device for the specified group.
dkepoint(3X)	dkgetepoint()	Retrieves the endpoint type code from the circuit associated with **fd** and stores the single character at **ep_type**. It is unchanged from previous releases.
dkepoint(3X)	dksetepoint()	Sets the endpoint type code for the circuit associated with **fd** to the single character stored at **ep_type**. It is unchanged from previous releases.
dkerr(3X)	dkerr()	Converts data switch error codes into the appropriate error messages for logging or printing. This function includes the new error codes of the newest releases of the AT&T data switches.
dkgos(3X)	dkget_goslevel()	Retrieves the channel transmitter GOS level. Only GOS3, GOS4, and the default value of GOS5 are supported. See the *dkgos* example later in this chapter.

Table 5-2: continued on next page

Table 5-2: *Continued*

Man Page	Function	Description
dkgos(3X)	dkset_goslevel()	Sets the channel transmitter to a different GOS level. Only GOS3, GOS4, and the default value of GOS5 are supported. See the *dkgos* example later in this chapter.
dkleveld(3X)	dkleveld()	Transmits a string of one or more URP level-D control codes or mixed data and control codes on an open CommKit Host Interface device. See the *dkleveld* example later in this chapter.
dkleveld(3X)	dkeof()	Transmits an URP level-D EOF code on an open CommKit Host Interface device. The EOF is scheduled for transmission in a separate URP block after any data already queued for the circuit.
dkleveld(3X)	dkbreak()	Transmits an URP level-D BREAK code on an open CommKit Host Interface device. The BREAK is scheduled for transmission in a separate URP block after any data already queued for the circuit.
dkleveld(3X)	dkusb()	Send a two-byte URP unsequenced data block on an open CommKit Host Interface device. The unsequenced data block is scheduled for the next transmission opportunity.
dkleveld(3X)	isdkleveld()	Tests whether the top message at the Stream head contains an URP level-D control code message. See the *isdkleveld* example later in this chapter.
dkleveld(3X)	isdkeof()	Tests whether the top message at the Stream head contains an URP level-D control code message with an EOF code as the first entry. See the *isdkeof* example later in this chapter.
dkleveld(3X)	isdkclosed()	Tests whether the circuit associated with the CommKit Host Interface device has been disconnected by the other end of the circuit. See the *isdkclosed* example later in this chapter.
dkmgr(3X)	dkmgr()	Establish the address as a SERVER on the AT&T Data Switch, using any interface or channel.
dkmgr(3X)	dknmgr()	Establish the address as a SERVER on the AT&T Data Switch, using any channel on the specified interface.
dkmgr(3X)	dknnmgr()	Establish the address as a SERVER on the AT&T Data Switch, using the specified channel on the specified interface.
dkmgr(3X)	dkmgrack()	Accept the incoming call to the SERVER.

Table 5-2: continued on next page

Table 5-2: *Continued*

Man Page	Function	Description
dkmgr(3X)	dkmgrnak()	Reject the incoming call to the SERVER.
dkmgr(3X)	dkinit()	Specify the URP initialization mode for the incoming call to the SERVER.
dkmgr(3X)	dkcktcfg()	Specify the URP window size for the incoming call to the SERVER.
dkmgr(3X)	dk_chk_idle()	Verify the specified interface and channel is IDLE and not in use on the system.
dkmgr(3X)	dk_reset_ckt()	Reset the specified interface and channel to the IDLE state.
dksplice(3X)	dksplice()	Splices together two existing circuits using the supplied file descriptors. **Note:** If the function call fails and the standard input was passed as one of the arguments to the function, the standard input must be re-initialized. See the *dksplice* example later in this chapter.
dksplwait(3X)	dksplwait()	Waits for an existing circuit to be spliced. Provides a timeout value so that a process may choose not to wait forever.
dktsplice(3X)	dktr_splice()	Requests a transparent splice of an incoming call to the specified address.
dktsplice(3X)	dktr_osplice()	Requests a transparent splice of an incoming call to the specified address within the specified channel range.
dktsplice(3X)	dktr_call()	Makes a call for transparent splice purposes setting the baud rate and URP parameters correctly for the spliced endpoints.
dkurpctl(3X)	dkset_no_ainit()	Sets the URP parameters to withhold acknowledgements of URP initialization requests.
dkurpctl(3X)	dkset_one_ainit()	Sets the URP parameters to respond to only one URP initialization request.
dkxenviron(3X)	dkxenviron()	Transmits local environment parameters to the remote system. This function is used by the remote execution interface.
maphost(3X)	maphost()	Maps partial dialstring information into a complete dialstring by searching the **dkhosts** table. It is unchanged from previous releases.

Table 5-2: continued on next page

Table 5-2: *Continued*

Man Page	Function	Description
maphost(3X)	miscfield()	Provides miscellaneous information from the **dkhosts** table. It is unchanged from previous releases.

System Call Compatibility

The current CommKit Host Interface software release supports the **open, close, read, poll**, and **write** system calls. The programmer should take into account that, with these two exceptions, the current CommKit Host Interface software release interfaces are STREAMS interfaces. The tty **ioctl** directives are supported by the standard UNIX system STREAMS module **ldterm**. Any low level support required by the **ldterm** module is supplied by the current CommKit Host Interface software release STREAMS module **dkty**. The **ioctl** interface supported in previous releases of the CommKit Host Interface software is no longer supported because the action:

- Is no longer needed because of architectural and implementation changes
- Is replaced by a library function or by an administrative command
- Is reserved for the internal use of the CommKit Host Interface software and is not supported for user-level applications code
- Has been dropped from this release.

The rest of this section gives a brief outline of the **open, close, read, write**, and **poll** system calls followed by a list of the old **ioctl** directives and their current disposition.

open(2)

The **open** system call to STREAMS drivers remains similar to the **open** call made to character drivers, with an expanded set of reasons – due to the exhaustion of STREAMS resources – for which the **open** call may fail. Application writers may wish to enhance their software to accommodate the new reason codes.

read(2)

The behavior of the **read** system call with STREAMS drivers is significantly different from character drivers. Some key points to note are:

Read Options The STREAMS **read** system call has three user-programmable modes:

- Byte-stream
- Message non-discard
- Message discard

Refer to the *UNIX System V Release 4 STREAMS Programmer's Guide.*

O_NODELAY When reading from a character driver, a device with no data will return a value of **0**. A Stream head will return a value of **-1** with *errno* set to EAGAIN.

LEVEL-D STREAMS **read** calls may return a **-1** with *errno* set to EBADMSG, because an URP level-D control code has arrived. Level-D codes arrive as M_PROTO messages and an application must use the appropriate CommKit Host Interface software library function such as **isdkleveld** to read the control code (refer to the *isdkleveld* example later in this chapter).

FAIL The **read** may fail because an error message such as an M_ERROR or an M_HANGUP reaches the Stream head.

write(2)

The **write** system call has changed. Some key points to note are:

BLOCKING A STREAMS device will only block on a write if it cannot accept data because of flow control directives by a lower level. Usually a **write** will return immediately. (See also O_NODELAY.)

O_NODELAY The behavior of a **write** to a STREAMS device with the O_NODELAY flag set is very different from a character device. Usually having the flag set will not make a difference since most writes to a STREAMS device return immediately. If, however, the devices cannot accept data then having the O_NODELAY flag set will cause the **write** to return a **-1** with *errno* set to EAGAIN. A **write** with O_NODELAY set may also return less than the requested amount if the device can

only accept part of the data.

LEVEL-D URP level-D codes cannot be written using the **write** system call since they have to be M_PROTO messages. The current CommKit Host Interface software release library routine **dkleveld** should be used to send URP level-D codes. Refer to the *dkleveld* example later in this chapter.

FAIL A **write** to a Stream head will fail, if it has received an M_ERROR message.

poll(2)

The **poll** system call may now be used with CommKit Host Interface STREAMS devices. It had been necessary for a user application to either wait for data to arrive (synchronous poll) or rely on some implementation-specific trick such as a signal to poll for data (asynchronous poll). CommKit Host Interface Release 3.2, for example, provides the DIOCSIG **ioctl** to signal a process when data has arrived. In STREAMS, the I_SETSIG STREAMS **ioctl** can be used with the **poll** system call to eliminate implementation-dependent tricks [see *streamio(7)* in the *UNIX System V Release 4 STREAMS Programmer's Guide*.]. Refer to the *poll* example later in this chapter.

close(2)

The **close** system call has changed the least but may cause certain applications difficulty if the timing required to close and proceed to the next activity is critical. The **close** system call in STREAMS waits up to 15 seconds, for each module and driver, to drain before the stream is dismantled and the system call returns. Applications which require special timing considerations should use O_NODELAY on the **open** system call. This will cause the **close** system call to return immediately, but may also cause other side effects with the **read** and **write** system calls.

dk_flush changes the **close** processing for flushing queued transmit user data on a data switch connection. *flush_time* may be specified to control how long **close** will wait for data to drain to the data switch's remote endpoint as one of the following:

-1 wait for all of the queued user data to drain.

0 compute a time delay based on the queued user data to drain; this assumes data drains at a rate of 10 bytes per second, equivalent to a 110 baud printer.

> wait the specified seconds for the queued user data to drain.

The waiting for the data to drain occurs when the user closes the **fd**, by means **close** or **exit**. When the queued data doesn't drain or the remote endpoint disconnects the circuit, **close** fails and returns an error to allow the user to determine the failure condition. This error return only occurs when **dk_flush** is called. Refer to the *dk_flush*(3X) manual page for return values.

ioctl(2)

The list of previously supported **ioctl** calls is given in Table 5-3 with a statement on the status of each.

Table 5-3: Previously Supported ioctl System Calls and Current Status

System Call	Status
DKIODIAL	All of the dial function is now provided by the expanded library function **dkitdial**. Refer to the *dkitdial* example later in this chapter.
DKIOCNEW	The announcement of a server name is now provided by the library function *dkmgr*.
DKIOCREQ	This function is not supported in user application code. All outgoing calls should use the **dkitdial** library routine. Refer to the *dkitdial* example later in this chapter.
DIOCRMODE	This *Set Receiver Mode* **ioctl** was used to select *Block Mode* or *Character Mode* and allowed the application to specify the timeout period for *Character Mode* within a 25 millisecond granularity. This **ioctl** is not supported by current CommKit Host Interface software release and the *Raw Driver Facility* receiver mode for a circuit is selected solely on the basis of the INIT0 or INIT1 control code received during circuit initialization. All incoming URP GOS1 and GOS2 data is available immediately and all URP GOS5 (block mode) data is available after the appropriate verification and processing. The timeout function may be re-implemented using a combination of the **poll** system call and the O_NODELAY **open** flag.
DIOCQQABO	The DIOCQQABO **ioctl** allowed the application to determine the completion status for the most recently completed **read** and was used in earlier implementa-

Table 5-3: continued on next page

Table 5-3: *Continued*

System Call	Status
	tions to determine why a **read** returned less than the requested bytes. Since the STREAMS environment disassociates the application's **read** from the **receive** performed on the network, it is no longer necessary to support this function. Application code should use the **dkleveld** functions to look for level-D or EOF indications. Refer to the *isdkclosed, isdkeof, isdkleveld* example later in the chapter. The current CommKit Host Interface software preserves message boundaries for received data, *except*: 1. It splits messages larger than 10240 bytes into smaller messages at boundaries convenient for the **dkhs** driver 2. It splits messages at every level-D URP code.
DIOCXCTL	In previous releases the only way an application could transmit an arbitrary URP level-D code was to use the DIOCXCTL **ioctl** directive. The specified level-D code would be transmitted at the end of the next data block followed by a possibly zero-length **write** to force the level-D code to be transmitted in an URP data block. This function has been replaced by the **dkleveld** library function. Refer to the *dkleveld* example later in this chapter.
DIOCSIG	An application could perform a form of asynchronous **read** with this **ioctl** request by "Posting a Receiver" on a circuit and being notified later with **signal** when the data arrived. This function is now provided through the STREAMS **poll** system call providing a cleaner more portable approach. Refer to the *poll* example later in the chapter.
DIOCBSIZE	This directive allowed the application to alter the URP transmit block size from the default value. This function is provided through the new library function **dk_info** which also allows the application to change the number of outstanding URP blocks. Refer to the *dk_info* example later in this chapter.
TCSBRK	The transmission of an URP level-D BREAK code was supported by the host driver through this **ioctl** in previous CommKit Host Interface implementations. This function is now provided through the **dkbreak** library function. The TTY subsystem continues to support the TCSBRK **ioctl**.
DIOOPEN	Certain DT/HYBRID features in the **dkserver** program used this **ioctl** to determine whether a specific channel on the interface was open. This function is not supported for user-level applications code.
DIOCINFO	Information about the calling channel and responsible interface could be obtained with this **ioctl** in previous releases of the CommKit Host Interface software. This

Table 5-3: continued on next page

Table 5-3: *Continued*

System Call	Status
	function is no longer supported.
DIOSRV	This **ioctl** was used to support the DT/HYBRID **-C** option for the **dkserver** program by checking whether the argument channel had any flags set in the channel control structure. This function is not supported for user-level applications code.
DIOCFLUSH	This directive would flush output queued for transmission on a circuit. This function is no longer necessary since the CommKit Host Interface software provides a 15-second drain period for all locally requested circuit terminations.
DKIOCSPN	This directive would splice together two circuits specified by Channel Number. This function is no longer supported. A similar function is provided by the **dksplice** library function.
DKIOCSPL	This directive would splice together two circuits specified by File Descriptor. This function is now provided by the **dksplice** library function.
DIOCSWAIT	This directive allowed an application to wait for its circuit to be spliced. This function is now provided by the **dksplwait** library function.
DKIORESET	This directive would allow a program to reset the entire interface. This function is provided by the **dkmaint** command.
DIOURPWD	This directive would allow a process to set both the URP transmit window and initialization mode for a circuit. A process may set the transmit window size by using the CommKit Host Interface library function **dk_info**. Processes no longer need to set the URP initialization mode.
DIOGETWD	This directive returned a copy of the host Receive Buffer Size. This function is no longer supported for user-level applications code.
DIOCQSTAT	This directive returned various interface error statistics. This function is supported by the **dkstat** command.

Examples

The code fragments shown in Figures 5-1 through 5-9 are examples of the new library functions. Some of the code is taken out of context from user-level commands. The intent of this section is to provide porting examples and is not to present a tutorial in application programming. Refer to the manual pages for more information.

dk_info Example

The example in Figure 5-1 is a self-contained program that shows the three commands of the **dk_info** library function.

Note: Retrieval and setting of the URP window parameters are not supported on the remote execution interface.

The following is a description of the **dk_info** library example, Figure 5-1:

- Lines 4 and 5 define storage for the two information structures needed.
- Lines 13 through 19 show the use of the DKGETIC command that returns the interface and channel number of the file descriptor for the standard input.
- Lines 21 through 28 show the use of the DKGETUW command that returns the parameters of the URP window.
- Lines 30 through 40 show the use of the DKSETUW command that sets the parameters of the URP window. It is important to set the *maxblocks* and *maxbytes* values before using this command.

If an error is returned by any of these calls there will also be an error message, printed on standard error, from the library function.

Figure 5-1: dk_info Example

```
 1      #include <stdio.h>
 2      #include <dkit/dk.h>
 3
 4      dk_intfchan_t myic;
 5      dk_urpwin_t myuw;
 6
 7      main() {
 8
 9              int fd;
10
11              fd = fileno( stdin );
12
13              if( dk_info(fd, DKGETIC, &myic) < 0 ) {
```

Figure 5-1: continued on next page

Figure 5-1: *Continued*

```
14                      fprintf(stderr, "dk_info DKGETIC failed \n" );
15              }
16      else {
20
17              fprintf(stderr,
18                      "intf=%d chan=%d \n", myic.iface, myic.chan);
19      }
21      if( dk_info(fd, DKGETUW, &myuw) < 0 ) {
22              fprintf(stderr, "dk_info DKGETUW failed \n" );
23      }
24      else {
25              fprintf(stderr,
26                      "maxblocks=%d maxbytes=%d \n",
27                              myuw.maxblocks, myuw.maxbytes);
28      }
29
30      myuw.maxblocks = 7;
31      myuw.maxbytes = 60;
32
33      if( dk_info(fd, DKSETUW, &myuw) < 0 ) {
34              fprintf(stderr, "dk_info DKSETUW failed \n" );
35      }
36      else {
37              fprintf(stderr,
38                      "maxblocks=%d maxbytes=%d \n",
39                              myuw.maxblocks, myuw.maxbytes);
40      }
41      return 0;
42  }
```

dkitdial Example

The code fragment in Figure 5-2 shows two examples of the **dkitdial** function. The following is a description of the code in Figure 5-2 that shows how the **dkitdial** function is used to select the interface of the outgoing call:

- Line 4 defines a dialstring and line 5 defines the needed structure.

- Lines 7–9 initialize the structure with the dialstring and interface number.
- Line 10 sets the command value to indicate that this is an ordinary dial (DK_DIAL) using the interface information (DK_SELINTF) set on line 8.
- Line 11 is the actual call to the function.

The following is a description of the code in Figure 5-2 that shows how the **dkitdial** function is used to select tty processing:

- Lines 13 to 15 initialize the structure but, because line 16 does not use the DK_SELINTF command, the functioning interface ignores the information and uses the default interface selection instead.
- Line 16 does use the DK_PUSHTTY command and this will request the function to **push** the tty processing modules onto the open stream.
- Line 17 is the actual call to the function.

Figure 5-2: dkitdial Example

```
 1      #include <dkit/dk.h>
 2
 3              int fdout0, fdout1, cmd;
 4              char dstring[128] = "ny/town/lake1";
 5              struct dkit_dial dl;
 6
 7              dl.dest = dstring;
 8              dl.intf = 1;
 9              dl.baud = (char *)NULL;
10              cmd = DK_DIAL|DK_SELINTF;
11              fdout1 = dkitdial(cmd, &dl);
12             /* make a call so tty processing is done./
13              dl.dest = dstring;
14              dl.intf = 0;
15              dl.baud = (char *)NULL;
16              cmd = DK_DIAL|DK_PUSHTTY;
17              fdout0 = dkitdial(cmd, &dl);
```

dk_namer Example

Figure 5-3 is an example of the **dk_namer** function. This function returns a pointer to a character string that is the name of the raw device indicated by the two arguments. The first argument (*intf*) is the interface number and the second argument (*chan*) is the data switch channel number on that interface.

Figure 5-3: dk_namer Example

```
1           char * dev_name;
2           int intf;
3           int chan;
4
5           /* get the device name for interface 0 and channel 112 */
6           intf = 0;
7           chan = 112;
8
9           dev_name = dk_namer( intf, chan );
```

dk_tnamer Example

Figure 5-4 is an example of the **dk_tnamer** function. This function returns a pointer to a character string that is the name of the tty device indicated by the two arguments. The first argument (*intf*) is the interface number and the second argument (*chan*) is the data switch channel number on that interface.

Figure 5-4: dk_tnamer Example

```
1   char * dev_tty_name;
2   int intf;
3   int chan;
4
5   /* get the tty device name for interface 0 and channel 112 */
6   intf = 0;
7   chan = 112;
8
9   dev_tty_name = dk_tnamer( intf, chan );
```

dk_xnamer Example

Figure 5-5 is an example of the **dk_xnamer** function. The function returns a pointer to a character string that is the name of the **xqt** device indicated by the two arguments. The first argument (*intf*) is the interface number and the second argument (*chan*) is the data switch channel number on that interface.

Figure 5-5: dk_xnamer Example

```
1   char * dev_xqt_name;
2   int intf;
3   int chan;
4
5   /* get the xqt device name for interface 0 and channel 112 */
6   intf = 0;
7   chan = 112;
8
9   dev_xqt_name = dk_xnamer( intf, chan );
```

dkgos Example

The example in Figure 5-6 is a self-contained program that shows the two commands within the **dkgos** library function.

Figure 5-6: dkgos Example

```
1       #include <errno.h>
2       #include <dkit/dk.h>
3       #include <stdio.h>
4
5       main()
6       {
7               int fd, goslevel, gos = GOS3;
8
9               if ( (fd = dkdial("nj/shore/vacation"))  < 0 ) {
10                      (void) fprintf(stderr, "Dkdial returned %d \n", fd);
11                      exit(2);
12              }
13
14              if ( ! dkget_goslevel(fd, &goslevel) ) {
15                      (void) fprintf(stderr,
16                              "Can't get gos level for fd %d \n", fd);
17                      exit(EIO);
18              }
19
20              if ( goslevel != gos ) {
21                      /* must set GOS transmitter to requested level */
22                      if ( ! dkset_goslevel(fd, &gos) ) {
23                              (void) fprintf(stderr,
24                              "Can't set fd %d gos level to GOS %d! \n",
25                              fd, gos);
26                              exit(EIO);
27                      }
28              }
29
30              close(fd);
31      }
```

Warning: ***Modifying the GOS levels should only be done by those who know and anticipate the effect on the application and network traffic.***

The following is a description of the *dkgos* library example, Figure 5-6:

- Lines 1 through 3 show the necessary *include* files.
- Line 7 defines the variables and sets the requested GOS level to GOS3.
- Line 9 uses the **dkdial** function to dial the host.
- Line 14 uses the **dkget_goslevel**() function to extract the current transmit GOS level and, upon success, stores the value in the *goslevel* variable. Failure of **dkget_goslevel**() causes Lines 15 through 17 to execute and the code exits.
- Line 20 checks whether the current GOS level matches the requested level. If not, line 22 uses the **dkset_goslevel**() function to set the GOS level to the requested level. If not successful, lines 23-26 report the error and exit.

dkleveld Example

Figure 5-7 demonstrates the use of the **dkleveld** function to send both data and level-D control codes. The program assumes that *stdout* is a data switch connection. Lines 6 through 12 define and initialize a **dk_lvld_t** message structure. The message contains a BREAK, three characters and another BREAK. The **dkleveld** function is called on line 21 and passed the file descriptor, the address of the message buffer and its length.

Figure 5-7: dkleveld Example

```
 1   #include <stdio.h>
 2   #include <sys/types.h>
 3   #include         <dkit/rdfp.h>
 4   #include <dkit/dk_urp.h>
 5
 6   dk_lvld_t        msg[] = {
 7                    dkDmkctl(URPdBREAK),
 8                    dkDmkdata('a'),
 9                    dkDmkdata('b'),
10                    dkDmkdata('c'),
11                    dkDmkctl(URPdBREAK)
12   };
13
14   main()
15   {
16            int fd, ret;
17
18            fd = fileno( stdout );
19            printf("start the test \n");
20            sleep(3);
21            ret=dkleveld(fd,msg,sizeof(msg));
22            printf("return code %d \n",ret);
23   }
```

dksplice Example

Figure 5-8 is an example of a simple splice recovery program and assumes that the original connection to the host was an ordinary tty login session.

The following is a description of the **dksplice** library example in Figure 5-8:

- Lines 1 through 3 show the necessary include files.
- Line 10 uses the **dkdial** function to dial the host to be spliced.
- Line 15 saves the tty settings of the standard input (file descriptor 0).
- Line 16 splices the call. If the splice works and the user is now connected to the new machine, the execution drops down to line 26, closes the open file descriptor, and exits. If the splice fails, the execution enters the recovery code at line 19.

- Lines 20 through 22 push the tty modules back onto the standard input stream.
- Line 23 resets the terminal modes.

Figure 5-8: dksplice Example

```
 1   #include <dkit/dk.h>
 2   #include <sys/stropts.h>
 3   #include <sys/termio.h>
 4
 5   main()
 6   {
 7            int rtn, fd;
 8            struct termio myterm;
 9
10            fd=dkdial("nj/shore/vacation");
11            if(fd<0){
12                     printf("dkdial returned %d \n", fd);
13                     exit(2);
14            }
15            ioctl(0, TCGETA, &myterm);
16            rtn=dksplice (fd,0);
17
18
19            if(rtn<0){
20                     ioctl(0, I_PUSH, "dkty");
21                     ioctl(0, I_PUSH, "ldterm");
22                     ioctl(0, I_PUSH, "ttcompat");
23                     ioctl(0, TCSETA, &myterm);
24            }
25
26            close(fd);
27   }
28
```

isdkclosed, isdkeof, and isdkleveld Example

Figure 5-9 shows the use of the three level-D query functions, **isdkclosed, isdkeof**, and **isdkleveld**. These functions replace the old DIOCQQABO **ioctl** directive which allowed the caller to find the status of the last **read** system call.

The function **vcs_read** uses the **read** system call, line 22, to read data from an open file descriptor *ds* of a data switch connection. The return value is checked on line 24 and, if it is greater than zero, the code breaks from the loop and returns the number of bytes read. Line 27 checks for returns less than zero.

Line 28 tests if *errno* is equal to EBADMSG. The EBADMSG indicates that the message at the head of the queue is not a data message.

Line 29 uses the **isdkeof** function to check for a level-D EOF code. If the function returns a true value then an EOF code has been received and the value of *nread* is set to EOF and the code breaks from the loop. If this is not the case, line 33 uses the **isdkleveld** function to check for any other level-D code. This function returns the level-D code or a zero. If there was a level-D code the loop continues and another **read** system call is executed.

Note: Since **isdkleveld** returns the level-D code, it could have been used to check for EOF by testing the return value with URPdEOF from the *dk_urp.h* header file.

For all other errors, the return code is set to the negative of the *errno* on line 37 and the code breaks the loop on line 38.

Line 40 checks for a zero return and calls **isdkclosed**. If the read returned zero bytes and the circuit is not closed, then this is a real zero-length read and the loop continues. If the circuit is closed the loop is broken on line 44 and a zero is returned.

Figure 5-9: isdkclosed, isdkeof, isdkleveld Example

```
 1          /*
 2          ** Read from VCS connection.
 3          ** Return value
 4          **      nread > 0,       number of bytes read.
 5          **      nread == 0,      circuit has been closed.
 6          **      nread == EOF,    level-D EOF received.
 7          **      nread < 0 (other than EOF),       nread is -errno.
 8          ** Note: Real zero length reads and level-D codes
 9          ** other that EOF are not returned by this function.
10          */
11  #include <stdio.h>
12  #include <errno.h>
13
14  vcs_read(ds)
15  {
16          static char      rbuf[BUFSIZ];
17          extern int       errno;
18          int nread;
19
20          for (;;) {
21
22                  nread = read(ds, rbuf, BUFSIZ);
23
24                  if( nread > 0 ) {
25                          break;
26                  }
27                  if( nread < 0 ) {
28                          if( errno == EBADMSG ) {
29                                  if ( isdkeof(ds) ) {
30                                          nread = EOF;
31                                          break;
32                                  }
33                                  if ( isdkleveld(ds) ) {
34                                          continue;
35                                  }
36                          }
37                          nread = -errno;
38                          break;
39                  }
40                  if( (nread == 0) && !(isdkclosed(ds)) ) {
41                          continue;
42                  }
```

Figure 5-9: continued on next page

Figure 5-9: *Continued*

```
43                      else {
44                              break;
45                      }
46              }
47              return nread;
48      }
```

poll Example

The code fragments in Figure 5-10 show how to poll a STREAMS file descriptor. This code replaces the old DIOCSIG **ioctl** directive for data switch circuits. The programmer should note that the STREAMS SIGPOLL signal is generated when the requested event happens, if it is the event at the head of the queue. If there are things already on the queue, you can use the **poll** system call with a zero timeout to look at the queue, as in the example.

Figure 5-10 assumes an open file descriptor *fd* to a data switch circuit and shows only the framework that would be placed around some local processing. Line 5 declares the signal catching function and lines 14 and 15 define a structure and a pointer used by the **poll** system call. Line 18 requests that the SIGPOLL signal be caught, line 27 is the **ioctl** call that requests that the Stream head send a SIGPOLL when ordinary data arrives. Once this request for service has been made it will be in effect until it is explicitly turned off.

Line 28 used the **poll** system call to look at one file descriptor. It is called with three arguments. The first is the address of the **pollfd** structure initialized on lines 20 and 21. The second is the number of **pollfd** structures found at this address; in this example it is one. The last argument is the timeout value which is set to zero so that the system call will not be blocked. Since we have only requested that **poll** look at the events on one circuit it will return a one if data has arrived on that circuit and a zero if there is no data available. If the **poll** is true then there will be some local processing, lines 34 to 41. Remember that if the **poll** is true there will not be any SIGPOLL signals unless the queue is emptied. If the **poll** test on line 28 is false, the example drops down to line 44 where there is more local processing. At the end of this local processing the example notifies the

Stream head to turn off the SIGPOLL notification on line 92. The example concludes with a simple signal catching function, starting at line 95. The programmer must remember to reset the SIGPOLL catching before reading all of the data off of the queue, since the default action of SIGPOLL is to kill the process. Since this fragment uses a very simple **signal** example and there are many new **signal** options in UNIX System V Release 4, we suggest anyone porting an application review all the new options.

Figure 5-10: poll Example

```
 1   #include <signal.h>
 2   #include <poll.h>
 3   #include <sys/stropts.h>
 4
 5   static void           catchpoll();
 6
 7            /*
 8            ** Do some processing but look for data on file
 9            ** descriptor 'fd'.
10            */
11   proc1(fd)
12   {
13
14            static struct pollfd      dkxpoll;
15            static struct pollfd      *fds = &dkxpoll;
16
17            /* arrange to catch SIGPOLL */
18            (void) signal(SIGPOLL, catchpoll);
19
20            fds->fd = fd;
21            fds->events = POLLIN;
22
23            /*
24            ** Watch for data on fd with the I_SETSIG ioctl.
25            */
26
27            ioctl(fd, I_SETSIG, S_RDNORM);
28            if ( poll( fds, 1 , 0 ) == 1 ) {
29                     /*
30                     ** If there is data on the stream from before the
31                     ** ioctl call, process it here.
32                     */
```

Figure 5-10: continued on next page

Figure 5-10: *Continued*

```
33-42
43    }
44
45    /*
46    ** Continue processing, if data arrives on fd this code will be
47    ** interrupted and the new data will be serviced by the signal
48    ** catching function. Remember that while we have I_SETSIG
49    ** active system calls may return -1 with errno set to EINTR.
50    */
51-87
88    /*
89    ** Stop watching for data on fd with the I_SETSIG ioctl.
90    */
91
92    ioctl(fd, I_SETSIG, 0);
93 }
94
95    static void
96 catchpoll(signo)
97 {
98    /* data has arrived on file descriptor fd */
99
100   /* reset the signal catching */
101   (void) signal(SIGPOLL, catchpoll);
102
103   /*
104   ** read and process new data
105   */
106-116
117 }
```

6 Manual Pages

NAME

dk – remote login or command execution via host interface

SYNOPSIS

dk [-j] *destination* [*command* [*args* ...]]

destination [*command* [*args* ...]]

DESCRIPTION

If no *command* is specified, the **dk** command sends a remote login request to the *destination* host. The *destination* host spawns a shell and the **dk** command logically connects it to the user's terminal. An AT&T data switch circuit is used to transfer data.

The remote shell bypasses the normal login sequence if you have been previously authorized access to the *destination* machine, through the use of *authorize*(1M). Instructions on the authorization process are found in the "Authorization" section.

The environment variables LOGNAME, HOME, and SHELL are initialized from */etc/passwd* on the *destination*. The PATH variable is initially set from the environment of the *dkserver*(1M) that accepts the call and may be modified by the user's profile. Other environment variables may be passed from the calling host to the *destination* host by listing them in the local environment variable DKEXPORT (for example, '**DKEXPORT=TERM,LINES,COLUMNS**'). The HOME directory is made the current directory.

The shell on the remote *destination* host is spawned with $0 set to '–**Dsh**' so that the files */etc/profile* and *$HOME/.profile* are executed before it prompts for any commands. The system accounting files are updated by *dkserver*(1M) for the benefit of *who*(1) and system accounting.

To exit from the remote shell, enter the Control-D sequence (or `exit`). If the program on the remote *destination* refuses to hang up, typing your QUIT character twice, rapidly, will break the connection from the calling end.

When invoked with the **-j** option, **dk** ignores job control for output control. This allows **dk** to run in the background, and therefore, run to completion when the user's shell provides job control capabilities (such as *ksh* and *jsh*).

NOTE: This may cause problems because the **-j** option allows the remote system to change the *stty* settings for the login terminal or, in terms of job control, the controlling terminal.

When invoked with a *command* argument to be remotely executed, **dk** causes a process to be spawned on the specified remote *destination* host and executes the given *command* there.

The actions taken are as previously described for remote login, except that the profiles are executed with $0 set to '–**Xsh**', no login accounting is done, and the program terminates after executing the single *command*.

dk may appear in a pipeline. Any use of standard input, standard output, or standard error on the remote *destination* is mapped into the same file descriptor locally. In particular, if standard output of the **dk** command is redirected, error messages will still be written to the terminal.

Linking the **dk** program to the names of popular hosts allows you to invoke the same commands by their *destination* names without using '**dk**'.

dk uses the file */etc/opt/dk/dkhosts* [see *dkhosts*(4)] to map the *destination* name to the appropriate data switch dialstring and to select the service to be invoked on that host. If the *destination* name contains no '**.**' or is not listed in *dkhosts*(4), **dk** will assume the '**rl**' or '**rx**' service for remote login or execution, respectively. To call hosts that don't support these services, or to call devices such as dialers in a modem pool, use the *dkcu*(1C) command. This could be done automatically by **dk** with the proper listing of options in *dkhosts*(4).

Authorization

In order to most effectively use this command, the user should be authorized on all hosts that will be used. This authorization process is accomplished using the command

dkauth *destination*

The remote *destination* will prompt the user for a `login` and `passwd` that are found in */etc/passwd* on the remote host. If the user answers correctly, the authorization service [see *authorize*(1M)] makes an entry in a translation file [see *dkuidtab*(4)] on the *destination* host that maps the originating host and user ID to a user ID on the remote host. Once the entry is made in *dkuidtab*(4), subsequent remote execution / login will bypass the login procedure on the remote *destination*.

Environment Variables

If the shell variable DKKEY is set in the user's environment, then that string is used as a matching token when authorizing. The token value is then used when mapping the originating host user ID to a user ID local to the *destination* host. For example,

DKKEY=*token* **dkauth** *destination*

The DKKEY value is stored in the *dkuidtab*(4) file on the remote *destination* host. Thus, from any given originating host and user ID, by changing the value of DKKEY, a user can remotely login as a number of different user IDs on a given remote host.

Multiple Interfaces

If multiple interface boards are installed on the originating host the *dk*(1C) command will use the default processing to select the interface for the outgoing call. See *dkauth*(1C) for more information.

EXAMPLES

The following command lines show the different ways that the **dk** command may be used. The **dk** command may be used for remote login specifying the dialstring of a remote data switch host as in:

dk wombat

where '**wombat**' is mapped into the appropriate data switch dialstring [see *dkhosts*(4)].

A valid data switch dialstring may be specified for remote login as in:

dk area1/exch1/host1.rl.vx

where **area1** is the *area,* **exch1** is the *exchange* and **host1** is the *destination.*

One can redirect the standard input, standard output, and standard error during remote execution using the **dk** command. For example,

dk wombat cat x 2>/dev/null

will redirect the standard error messages to */dev/null.*

Whenever you login from one machine to another, or remotely execute a command, your *.profile* is read to set up your environment the way you wish, so the invoked command will get the right PATH, *umask*, etc. You should distinguish between the different cases by the name of the invoked shell as it was stored in the variable $0. The value of $0 will be '–**sh**' or '–**ksh**' for a login from 'DESTINATION:', '–**Dsh**' for remote login via the **dk** command, and '–**Xsh**' for remote execution.

Since the user's *.profile* is executed on the remote machine during remote execution, you should never do any prompting for terminal type (for example, if **$0 == '–Xsh'**) and you might not want to set tabs, for instance, if **$0 == '–Dsh'**. Shell variables such as TERM may be exported across system boundaries by setting DKEXPORT.

Warning: When **$0 == '–Xsh'** the profile should **never** exec another program (as in '**exec ksh**') or run any program that requests standard input from the terminal or sends standard output or error to the terminal. If you fail to abide by this rule, your attempts at remote execution may not be successful.

A sample which illustrates the proper setup of the *.profile* follows:

```
HOME='pwd'
umask 022
PATH=$PATH:/opt/dk/bin:/bin:$HOME/bin:/usr/bin:/usr/ucb
DKEXPORT=TERM
```

```
export PATH CDPATH HOME MAIL PS1 TERM DKEXPORT
MAIL=/usr/mail/$LOGNAME
EDITOR=vi ED=vi MAILER=mail
export EDITOR ED MAILER MAIL
PS1="HELLO>"
case $0
in
   -Dsh)
      # remote login
      echo "you are on `uname`"
      trap `echo logged off `uname`` 0
      mesg n
      ;;
   -Xsh)
      # remote execution
      #
      # NEVER prompt for anything or `exec` another
      # program from this case.  If you perform an
      # `exec ksh` when $0 == `-Xsh`, remote execution
      # will not work.
      ;;
   -sh | -ksh)
      # regular login
      if [ "`tty`" != "/dev/console" -o x${TERM} = "xansi" ]
      then
          echo "TERM=\c"
          read TERM
      fi
      mesg n
      stty tabs erase ´^h´ kill ´^o´ echoe cr0
      tabs
      date
      ;;
esac
```

UNIX poll(2) command is supported in remote execution driver. This will enable applications like *EMACS* to work over *dk* command. Refer to manual page on *poll*(2) for further details on this feature usability.

FILES

/opt/dk/bin	directory in which this command resides
/etc/opt/dk/dkhosts	host control file for destination mapping
$HOME/.dkhosts	local control file for destination mapping
/etc/opt/dk/dkuidtab	default user ID mapping file
/dev/dkx/*intf***.***chan*	remote device names
/etc/profile	

SEE ALSO

dkauth(1C), *dkcu*(1C), *authorize*(1M), *dkserver*(1M), *dkdial*(3X), *maphost*(3X), *dkhosts*(4), *dkuidtab*(4).
cat(1), *login*(1), *passwd*(1), *who*(1), *poll*(2) in the *UNIX System V User's Reference Manual*.

WARNINGS

The protocol used by remote login and remote execution has a large overhead and users may notice that things appear to be much slower than a login session from a 'DESTINATION:' prompt or one initiated from a *dkcu*(1C) command. Single character output, in particular, is very inefficient, and for programs that don't buffer their terminal output, piping the results through *cat*(1) will often speed up output.

The user's profiles must not read nor write anything to the standard output when invoked for remote execution (where **$0 == '–Xsh'**), or services based **dk.**

NAME

dkauth - manage dkuidtab entries and authorize to remote hosts

SYNOPSIS

dkauth [– **a**] [– **i** *intf_list* | – **g** *group*] *area/exchange/host*
dkauth – **d** [– **u** *uidfile*] [– **l** *login*]
dkauth – **x** [– **u** *uidfile*] [– **l** *login*] [*originating_grp* [*.uid*] [*/DKKEY*]]
dkauth – **r** [– **u** *uidfile*] [– **l** *login*] [*originating_grp* [*.uid*] [*/DKKEY*]]

DESCRIPTION

When invoked as a user-level command on the local host **dkauth** can:

1. Perform an authorize using the specified interfaces to the address, or
2. Display, delete, or rejuvenate the user's *dkuidtab* entries.

Authorize to Remote Hosts

The **dkauth** options used to authorize a user to a remote host are:

-a The *-a* option will perform an authorize using the specified interfaces to the address. In the absence of any option, **dkauth** assumes the *-a* option.

-i The *-i* option specifies a comma-separated interface list (or the special value *all* to try all possible interfaces) for placing calls. The *-i* option overrides the value of DKINTF. The *-i* option cannot be used with the *-g* option.

-g The *-g* option specifies a channel group defined in the *dkgroups* file. The *-g* option cannot be used with the *-i* option.

The shell variable DKKEY can be used with **dkauth**.

On invoking the command:

dkauth -a host

The user is prompted for a remote login and password. Answering the login ID prompt with a carriage return will cause the current authorization (if any) to be deleted on the destination host.

When multiple interfaces are selected, authorization attempts are made on all interfaces, even when a connection cannot be established. An error message from **dkdial**(3X) is displayed detailing the reason for the connection failure.

Environment Variables

The environment variable, DKKEY, can be specified on the command line, however, if it is *not* specified, the following description applies.

If the shell variable, DKKEY, is set in the user's environment, then that string is used as a matching token when authorizing. The token value is used when mapping the originating host user ID to a user ID local to the *destination* host. For example,

DKKEY=*token* **dkauth** *destination*

The DKKEY value is also stored in the *dkuidtab*(4) file on the remote *destination*.

Thus, from any given originating host and user ID, by changing the value of DKKEY, a user can remotely login as one of a number of different user ID's on a given *destination* host.

Multiple Interfaces

The *CommKit* Host Interface software supports multiple interface boards, allowing connectivity to multiple data switch networks or redundant connectivity to a single network. If the interfaces reside in different data switch *Originating Groups*, an authorization made from one interface will not be valid for calls established through one of the other interfaces. The user can invoke the *-i* option to authorize over selected or all interfaces. For example:

dkauth -a -i 0,1,3 nj/casino/slots

authorizes the current user on interfaces **0,1**, and **3** to **nj/casino/slots**.

In the absence of the *-i* option, the command allows interface selection by using the DKINTF environment variable. When DKINTF is not set, any one of the interfaces is used.

For example, on a host with two interfaces connected to the same data switch, a user can authorize from the originating machine to the *destination* machine through both interfaces with the following command:

DKINTF=0,1 **dkauth** *destination*

The DKINTF variable may be set and exported in the *.profile* file or by shell scripts if the interface preference is required across several connection attempts.

Manage *dkuidtab* Entries

Invoking **dkauth** with the **-d, -x,** or **-r** option allows the user to manage entries in the *dkuidtab* file. The following options are supported with the **-d, -x,** and **-r** options:

-l *login* This option specifies the local user ID (other than the default ID) for which mapping information is desired. The default is the user ID of the user executing the command. Only *root* may specify any *login* other than its own. All other users may only specify their own *login*.

– **u** *uidfile* This specifies the *dkuidtab* file which **dkauth** should use to get the user ID mapping information. The default file is */etc/opt/dk/dkuidtab*.

Display Using the *-d* option will display the user's *dkuidtab* entries. By default, the current user's *dkuidtab* entries are displayed, however, **root** can display the *dkuidtab* entries for all users by specifying *all* as the login name.

Delete The *-x* option is used to delete the user's *dkuidtab* entries. By default, *-x* deletes all *dkuidtab* entries for the specified user. A user can control which entries to delete by specifying an originating group with an optional *.uid* and */DKKEY* field. The command will delete the *dkuidtab* entries based on all fields; if the *uid* or *DKKEY* option is missing, that field will be ignored

for matching criteria.

Rejuvenate The *-r* option rejuvenates the user's *dkuidtab* entries. By default, *-r* updates all entries for the specified user. A user can change which entries to rejuvenate by specifying an originating group with an optional *.uid* and/or */DKKEY* field. The command will rejuvenate the *dkuidtab* entries based on all fields; if the *uid* or *DKKEY* option is missing, that field will be ignored for matching criteria.

Whenever the *passwd*(4) entry is changed on the *destination* host, the user must rejuvenate authorization service to update *dkuidtab*(4).

EXAMPLES

Remote Authorization

The following example shows a remote authorization to a system with the old *authorize* service. This example uses the default selection of the interface with no interface information displayed.

```
dkauth nj/casino/slots
Enter Remote login id: player
Enter Remote Password:
You are now authorized as player on nj/casino/slots
```

The next example shows a request to authorize on all interfaces with a failure on one of the interfaces (specifically, interface 2).

```
dkauth -iall nj/casino/slots
Enter Remote login id: player
Enter Remote Password:
Intf 0: You are now authorized as player on nj/casino/slots
Intf 1: You are now authorized as player on nj/casino/slots
dkdial: Can't open /dev/dk/dial2, errno 19: No such device
Intf 3: You are now authorized as player on nj/casino/slots
```

The next example shows a request to authorize on selected interfaces.

```
dkauth -i 0,3 nj/casino/showboat
Enter Remote login id: star
Enter Remote Password:
Intf 0: You are now authorized as star on nj/casino/showboat
Intf 3: You are now authorized as star on nj/casino/showboat
```

The next example shows an authorization on a remote system on a specific interface being removed by means of entering a carriage return at the *Login* prompt.

```
dkauth -i 0 nj/casino/showboat
Enter Remote login id:
Intf 0: Your entry on nj/casino/showboat from nj/shore/bird.1939 has been deleted.
```

The following example shows a remote authorization to a system with the new *authorize* service:

```
dkauth nj/casino/showboat
Enter Remote login id: captain
Enter Remote Password:
You are now authorized as captain on nj/casino/showboat from nj/shore/bird.1939
```

The above example, using the new *authorize* service displays origination information (*from nj/shore/bird.1939*).

Display Authorization

As an example, the command:

```
dkauth  -d -l  major

     USER major ON SYSTEM post HAS BEEN AUTHORIZED FROM:

       area/exch/group             uid#
----------------------             ------
       nj/casino/slots             12  authorization expired
    nj/casino/showboat             345
       nj/casino/ocean             67/defg
```

will locally invoke **dkauth**. The above display is based on the assumptions:

1. This command is executed by *root* or user *major* on host *post*
2. User ID 12 on *nj/casino/slots* has been authorized/mapped to user *major* on host *post*, but *major*'s password on *post* has been changed since the authorization
3. User ID 345 on *nj/casino/showboat* has been authorized/mapped to user *major* on host *post*
4. User ID 67 on *nj/casino/ocean* has been authorized/mapped to user *major* on host *post* with DKKEY=defg.

Any display line followed by **authorization expired** means that authorization is no longer valid for one of the following reasons:

1. The password for the user has expired,
2. The password for the user has been changed.

This is an indication that the user for whom the authorization check was done (user *major* in the example) needs to re-authorize to the local host (*post*) from the specified remote host (*slots*).

Delete Authorization

Consider the following examples of the *-x* option:

```
dkauth -x nj/casino/slots
dkauth -d -l major


USER major ON SYSTEM post HAS BEEN AUTHORIZED FROM:

             area/exch/group             uid#
   -------------------------             ------
          nj/casino/showboat             345
             nj/casino/ocean             67/defg
```

The above example has deleted all entries from the group **nj/casino/slots**.

```
dkauth -x nj/casino/showboat.354
dkauth -d -l major

 USER major ON SYSTEM post HAS BEEN AUTHORIZED FROM:

   area/exch/group             uid#
------------------------       ------
   nj/casino/slots             12  authorization expired
nj/casino/showboat             345
   nj/casino/ocean             67/defg
```

This example has not deleted any entries as there were none with a group uid of **354**.

```
dkauth -x nj/casino/ocean/defg
dkauth -d -l major

 USER major ON SYSTEM post HAS BEEN AUTHORIZED FROM:

   area/exch/group             uid#
------------------------       ------
   nj/casino/slots             12  authorization expired
nj/casino/showboat             345
```

This example has deleted all entries with the DKKEY value **defg**.

Rejuvenate Authorization

Consider the following example of the *-r* command:

```
dkauth -r nj/casino/slots
dkauth -d -l major
```

```
USER major ON SYSTEM post HAS BEEN AUTHORIZED FROM:

          area/exch/group             uid#
------------------------              ------
          nj/casino/slots             12
       nj/casino/showboat             345
          nj/casino/ocean             67/defg
```

FILES

/etc/opt/dk/dkgroups	channel group control file
/etc/opt/dk/dkuidtab	default user ID mapping file
/etc/opt/dk/dkuidtab:o	backup copy of the user ID mapping file
/etc/passwd	password file
/etc/shadow	encoded password file

SEE ALSO

authorize(1M), *dkserver*(1M), *dkdial*(3X), *dkgroups*(4), *dkuidtab*(4).
passwd(1) in the *UNIX System V User's Reference Manual*.

DIAGNOSTICS

When a user attempts to execute the **dkauth** command, several error messages may result. The exit code is zero for normal exit, otherwise it is one of the following types from *dkit/sysexits.h*:

EX_USAGE	Usage errors.
EX_SOFTWARE	Internal processing error occurred (invalid process_option).
EX_DATAERR	Invalid interface value in DKINTF. Unknown loginid in */etc/passwd* from *-l login*.
EX_OSERR	The expected UNIX system operation failed.
EX_OSFILE	Invalid *dkuidtab* entry *uidfile* update failed.
EX_NOPERM	Attempting to access a *loginid* other than user's, when not the super-user.

dkdial error messages and **dkdial** return codes are documented in *dkdial*(3X).

All error messages are preceded by the string <*Progname*>:. Most of those messages are documented below.

Unknown login '<*login*>**'**

The specified *loginid* is invalid.

Permission denied to examine '<*loginid*>**'**

A user other than *root* is not permitted to specify a *loginid* other than her/his own with the *-l* option.

Permission denied to examine all entries
Only the super user may specify *-l all*.

Cannot access uidfile '<*uidfile*>**',** <*UNIX error*>

The specified *uidfile* does not exist or could not be opened. *UNIX_error* gives the reason.

Cannot specify a <*option*> **when doing remote authorization**
The user specified a *-u* or *-l* option either with no other options or with the *-a* option.

Cannot specify interfaces when <*action*> **authorization information**
The user specified a *-i* option with one of the following *actions*: displaying *-d*, rejuvenating *-r*, deleting *-x*.

uid <*uidnum*> **not found in password file**
The user didn't specify the *-l* option with the *-d*, *-r*, or *-x* options. When the command attempted to convert the current user's UID to a loginid it failed.

Could not open uidfile '<*uidfile*>**' for reading,** <*UNIX_error*>
The named *uidfile* could not be accessed to read the data. The <*UNIX_error*> describes the UNIX error for the failed access.

Uidfile '<*uidfile*>**' is corrupted.**
Please notify the System Administrator.
The named *uidfile* contains invalid entries. If this is not a test *uidfile*, the UNIX system administrator should be notified.

Cannot change uid to correct value, <*UNIX_error*>
The *uid* cannot be processed.

DKINTF contains invalid interface value: '<*ivalue*>**'**
The *ivalue* is not defined for this system.

Failed to get login id, <*UNIX_error*>
The command failed to read the user's remote login id because of the <*UNIX_error*>.

Intf <**inum**>**:** <*Authorization error messages*>
Any message other than those listed above are output from the authorize service from the remote system. See *authorize*(1M) for details of these messages.

A variety or error messages relating to updating the *dkuidtab* file may be generated when using the *-d*, *-x*, and *-r* options. For example:

Cannot create uid update file '<*tmpfile*>**',** <*UNIX_error*>
When attempting to create the *tmpfile* to update the dkuidtab, the file creation failed for the specified <*UNIX_error*>.

Login information for '<*loginid*>**' not available.**
When attempting to get the *loginid*'s updated password the password entry was not available. This is probably a problem with */etc/passwd* and / or */etc/shadow*.

Cannot create temp file '<*tmpname*>**',** <*UNIX_error*>
The command cannot create a temporary file, <*tmpname*>, for processing of the option, *-l all*. The value of <*UNIX_error*> defines the problem with creating the file.

In addition, a variety of error messages dealing with a failure of the *dialstring* may be generated. For example:

Intf <inum>: Login prompt from <*dialstring*> **failed,** <*UNIX_error*>
Reading of the login prompt from *dialstring* failed because of <*UNIX_error*>. The connection to the remote system was on the interface specified by *inum*; if no interface was specified this information is not displayed.

NAME

dkcat – concatenate files and send them to a non-host AT&T data switch destination

SYNOPSIS

dkcat [– **I**] [– **F**] [– **q** *qc,ec*] *destination* [*files* ...]

DESCRIPTION

dkcat transmits files to a non-host *destination*, such as a printer, attached to your data switch network. **dkcat** places a call to *destination* and transmits the *files*. **dkcat** waits for all file data to be received at the destination before terminating.

The following options are supported by **dkcat**:

– **F** Send a form-feed character after each file. This is the *only* option available for garden-variety printers.

– **I** Selects Imagen-specific processing.

– **q** *qc,ec* Specifies a "quote character" *qc* and an "EOF character" *ec* to override the defaults of **155,158**. The character definitions must be in decimal format separated with a comma, with no spaces between the definitions.

destination If the *destination* is not a full network address, i.e., *area/exchange/end-point*, then **dkcat** will attempt to map it to a full address using the file */etc/opt/dk/dkhosts* and a service *class* of **p** (for printer).

files Standard input is read if no *files* are named or when a file name of "– " is encountered.

Multiple Interfaces

If multiple interface boards are installed on the originating host, **dkcat** will use the default processing to select the interface for the out-going call. See *dkauth*(1C) for more information.

FILES

/opt/dk/bin	directory in which this command resides
/etc/opt/dk/dkhosts	host control file for destination mapping
$HOME/.dkhosts	local host control file for destination mapping

SEE ALSO

dkauth(1C), *dk_flush*(3X), *dkdial*(3X), *maphost*(3X), *dkhosts*(4).

NAME

dkcu – call another host

SYNOPSIS

dkcu [– **s**] [– **f**] [– **d**] [– **r**] [– **v**] [– **w**] [– **x**] [– **b 7** | **8**] *destination*

DESCRIPTION

dkcu dials another UNIX System, a terminal, or possibly a non-UNIX System. It manages an interactive conversation with possible transfers of ASCII files.

It places a call to the *destination* host or terminal on the AT&T data switch network. Several options are supported by **dkcu**:

– **s** Suppresses the "Circuit Open" and other non-error messages.

– **f** Forces a **dkcu** even if the user came in as a remote executor.

– **d** Used to get tracing and diagnostic output.

– **v** Local environment variables may be passed from the calling host to the *destination* host by listing them in the local environment variable DKEXPORT (such as, **DKEXPORT=TERM,LINES,COLUMNS**). When using this option, the *destination* should be appended by **rl** and **vt** flags (such as, **dkcu -v** *destination***.rl.vt**) and the user should be authorized [see *authorize*(1M)] on the *destination* host.

– **w** Wait until an end of file (e.g., hang up) condition on the conversation before terminating the **dkcu** session. By default, **dkcu** immediately terminates when its TTY standard input file descriptor returns an end of file condition. When interacting with an operator at a TTY device, or when **dkcu** is being executed in background, this is often the preferred behavior. However, when **dkcu** TTY standard input is coupled to a shell script or other program through a pipe, this behavior has the side effect of stopping the **dkcu** session to the remote end point before all of the output from that end point has been displayed or processed. The **-w** command-line option forces **dkcu** to wait for an end of file condition on its conversation before exiting. This allows all of the conversation's data from the remote end point to drain to **dkcu** standard output. In the example below, the **-w** command line option is used to prevent **dkcu** from exiting prematurely:

```
cat script_file | dkcu -w xmpladrs | myprogrm
```

– **x** Requests that XON/XOFF output flow control be done locally; otherwise, XON/XOFF characters are passed through to the *destination*.

– **r** Turn off all "~" processing except "**~ .**" and "**~ %break**." When using the – **r** option, all other lines beginning with " ~ " will return a restricted mode error message.

– **b bits**
Used to force the number of bits per character processed on the connection. Valid values for bits are 7 and 8. The default is 7 bit characters unless the terminal is set for 8 bit characters *and* no stripping of input characters to 7 bits.

After making the connection, **dkcu** runs as two processes: the *transmit* process reads data from standard input and, except for lines beginning with tilde (~), passes it to the remote system. The *receive* process accepts data from the remote system and, except for lines beginning with ~, passes it to standard output. Lines beginning with ~ have special meanings.

The *transmit* process interprets the following:

~ .	Terminate the conversation. If the program on the remote host isn't reading input, typing the QUIT character twice, rapidly, will break the connection.
~ !	Escape to an interactive shell on the local system.
~ !*cmd*...	Run *cmd* on the local system (via **sh** – **c**).
~ $*cmd*...	Run *cmd* locally and send its output as standard input to the remote system.
~ %cd [*dir*]	Change directory to $HOME or *dir* on the local system. Note: **~ !cd** will cause the command to be run by a sub-shell, probably not what was intended.
~ %take *from* [*to*]	Copy file *from* (on the remote system) to file *to* on the local system. If *to* is omitted, the *from* argument is used in both places.
~ %put *from* [*to*]	Copy file *from* (on local system) to file *to* on remote system. If *to* is omitted, the *from* argument is used in both places. Permission to create or overwrite the *to* file must be allowed.
~ %break	Transmit a BREAK to the remote system (which can also be specified as **~ %b**).
~ %debug	Toggles the -d debugging option on or off (which can also be specified as **%d**).
~ %divert	Toggles unsolicited diversion (enabled / disabled) to files. When unsolicited diversion is enabled, new-style diversion from standard output to a file is initiated by **~ [local]**>:*filename*, and is terminated by **~ [local]**>, where *local* is the nodename of the local system. This enables **~ %take** to operate properly when **dkcu** is used over multiple hops.
~ %old	Toggles old-style diversion syntax (enabled / disabled). When unsolicited diversion is enabled, old-style diversion uses ~ >:*filename* to initiate output diversion and ~ > to terminate it. Enabling old-style diversion does not prevent new-style diversion from working.

~ t Prints the values of the termio structure variables for the user's terminal (useful for debugging). See *termio*(4) for more information.

~ ~... Send the line "~..." to the remote system.

The use of **~ %put** requires *stty*(1) and *cat*(1) on the remote side. It also requires that the current erase and kill characters on the remote system be identical to the current ones on the local system. Backslashes are inserted at appropriate places.

The use of **~ %take** requires the existence of *echo*(1) and *cat*(1) on the remote system. Also, **stty tabs** mode should be set on the remote system if tabs are to be copied without expansion.

When **~ %take** is used to transfer files, the filename cannot exceed the length imposed by the UNIX system. This length limit is discussed in the *termio(7)* manual page. Additionally, the values MAX_CANON and MAX_INPUT defined in */usr/include/sys/param.h* and */usr/include/limits.h* affect the filename length. The shell uses a complex **if** statement to determine if the named file in the **~ %take** transfer exists; approximately 180 characters should be provided in the buffer for this **if** statement. Therefore, the actual filename size should be these values (MAX_CANON, MAX_INPUT) minus 180.

When **dkcu** is used on system A to connect to system B and subsequently used on system B to connect to system C, commands on system B can be executed by using ~ ~. Executing a tilde command reminds the user of the local system uname. For example, uname can be executed on C, A, and B as follows:

```
uname
C
~[A]!uname
A
~ ~[B]!uname
B
```

Note that [A] and [B] are output by **dkcu**, not typed by the user. In general, ~ causes the command to be executed on the original machine, while ~ ~ causes the command to be executed on the next machine in the chain.

The suspend character (usually **^Z**) puts the current foreground process of the remote system in the background. The **dkcu** process of the local system can be suspended (placed in the background) with **~ ^Z**.

Multiple Interfaces

If multiple interface boards are installed on the originating host, **dkcu** will use the default processing to select the interface for the outgoing call. See *dkdial*(3X) and *authorize*(1M) for more information.

FILES

/opt/dk/bin directory in which this command resides

SEE ALSO

pull(1C), *push*(1C), *authorize*(1M), *dkdial*(3X).
cat(1), *echo*(1), *stty*(1), *cu*(1C), *uucp*(1C), *termio*(4), *termio*(7) in the *UNIX System V User's Reference Manual*.

DIAGNOSTICS

Exit code is zero for normal termination initiated with ~. on the local host, non-zero otherwise.

WARNINGS

dkcu should be used with the **-x** option on terminals that are directly connected to a host.

NOTES

dkcu buffers input internally.

There is an artificial slowing of transmission by **dkcu** during the **~ %put** operation so that loss of data is unlikely. If the *to* file of **~% put** cannot be created, an error message will be displayed, but the *from* file will then be written to */dev/null*.

NAME

dkdo – transparent remote execution facility

SYNOPSIS

dkdo [– **f** *controltable*] *command* [*args* ...]

command [*args* ...]

DESCRIPTION

dkdo provides a generalized, transparent remote execution facility with automatic file transfer. One use of **dkdo** is to emulate a command such as *lp*(1) requiring facilities like line printers or RJE support not present on the local machine. It works by placing a call through the AT&T data switch network to a host that does have the required facilities, sending any input file arguments to the remote system, executing the command there, then bringing back any output files.

The input files are placed in a new directory created under */tmp* on the remote machine which is used as the current directory when the command is invoked. Any files created in that directory by the command are brought back to the current directory on the local machine upon completion. The remote command is executed with the same standard input, output, and error file descriptors as on the local machine.

The function of the command is controlled by a *controltable* which describes how the commands are to be executed and the format of the arguments. If not specified by the '**-f**' argument, the default file */etc/opt/dk/dkdotab* is used. The format of the *controltable* is described in the manual page for *dkdotab*(4).

Invoking the program by the name **dkdo** is the explicit invocation format. Usually, however, the second, implicit format is used by linking the **dkdo** program to *lp*(1) or another program. Then users may use the emulated command as if it were actually located on the local machine. In this case the link name, that is, *lp*(1) is the *command* that will be remotely executed.

Multiple Interfaces

If multiple interface boards are installed on the originating host, then **dkdo** will use the default processing to select the interface for the outgoing call. See *dkauth*(1C) for more information.

FILES

/opt/dk/bin	directory in which this command resides
/etc/opt/dk/dkhosts	host control file for destination mapping
$HOME/.dkhosts	local host control file for destination mapping
/etc/opt/dk/dkdotab	default **dkdo** control table

SEE ALSO

dkauth(1C), *maphost*(3X), *dkdotab*(4), *srvtab*(4).
lp(1) in the *UNIX System V User's Reference Manual.*

DIAGNOSTICS

The **dkdo** command will fail on the remote machine if there is no space left in the file system containing */tmp.* One of the following error messages will appear on the user's terminal on the local machine:

```
mkdir:  Failed to make directory "dkdo.xyz";
              No space left on device
```

```
dkpp: cannot link filename
```

```
dkpp: filename extract write error
```

where *filename* is the full pathname of the file to be transferred to/from the remote/local host computer. Lack of space on the remote is **not** the only case where these error messages may appear.

NAME

pull – transfer files from another system

SYNOPSIS

pull [**-L**] *destination filename ... directory*

DESCRIPTION

pull establishes an AT&T data switch circuit to a source (remote) host named in *destination* and transfers files from that host.

The *filename*s are files or directories on the source machine and are interpreted relative to the user's HOME directory on the source *destination* if they do not begin with a ´/´. The *filename*s are placed in the *directory* on the target (local) machine. If the *directory* does not begin with a ´/´ it is interpreted relative to the current directory. The *directory* will be created, if required, before the files are transferred. Specifying a directory as one of the *filename*s will transfer the entire directory tree beginning at the named point.

pull preserves the file modes and modification times of the files it moves. The original file owner (numeric user ID) is preserved if the effective user ID of the process on the local machine is **root**; otherwise, the files will be owned by the current user.

pull preserves the name, type and contents of the files it moves with the following exceptions.

Long Names

If the target file system does not support file names greater than 14 characters and the source machine transfers a file with a name greater then 14 characters, the name will be truncated to 14 characters. The local side will warn the user for each file name that is truncated.

Symbolic Links

Files of type symbolic link will be preserved unless the '– **L**' option is used. A transferred symbolic link will be identical to the source file. It may, however, have a different context in the target environment. If the '– **L**' option is used, the source machine will be asked to follow symbolic links with the following behaviors: the symbolic link will be treated as if it were the file type of the file pointed to. If the file does not exist or the user does not have permission to access the file, no transfer will take place and the source machine will issue a warning. If, in following a path that contains a symbolic link that points to a directory, the source machine finds a directory that has already been transferred, it will not transfer it a second time. The source however, will always transfer directories in a path that does not contain a symbolic link even if that directory has already been transferred while following a path that does contain a symbolic link. In both cases, the source machine will warn the user for

every affected directory.

The *push*(1C) and **pull** commands invoke the **pupu** program (*/opt/dk/bin/pupu*) on the remote system to handle the remote end of all file transfers. If the *srvtab*(4) file on the remote system invokes the **pupu** program with the ´– **r**´ option, file transfers will be restricted to/from the home directory tree on the remote system by disallowing all paths that begin with a leading ´/´ or that contain an embedded ´**..**´ specification. In this restricted mode, only transfers containing PATHs relative to the HOME directory will be accepted.

Multiple Interfaces

If multiple interface boards are installed on the originating host, the **pull** command will use the default processing to select the interface for the out-going call. See *dkdial*(3X) and *authorize*(1M) for more information.

WARNINGS

If the source *destination* is identical to the target host and the source *directory* is identical to the target directory, then the **pull** command will overwrite *filename* and its contents may be destroyed.

The modification times are preserved on transferred files as long as they are earlier in *relative* time (i.e., seconds since the 00:00:00 GMT, January 1, 1970, epoch) than the current relative time on the local machine. If the modification time for a file would date that file in the future on the local machine, the current time on the local machine will be used as the file modification time.

FILES

/opt/dk/bin	directory in which this command resides
/etc/opt/dk/dkhosts	host control file for destination mapping
$HOME/.dkhosts	local host control file for destination mapping

SEE ALSO

push(1C), *authorize*(1M), *dkdial*(3X), *maphost*(3X), *dkhosts*(4), *srvtab*(4).
time(2) in the *UNIX System V Programmer's Reference Manual.*

NAME

push – transfer files to another system

SYNOPSIS

push [– **L**] *destination filename ... directory*

push [– **L**] *destination – directory < file_list*

DESCRIPTION

push establishes an AT&T data switch circuit to the target (remote) host named in *destination* and transfers files to that host.

The *filename*s are files or directories on the source (local) machine. The *filename*s are placed in the *directory* on the target machine. If the *directory* does not begin with a ´/´, it is interpreted relative to the user's HOME directory on the target *destination*. The *directory* will be created, if required, before the files are transferred. Specifying a directory as one of the *filename*s will transfer the entire directory tree beginning at the named point.

The second command format takes the list of files to transfer from the standard input. It differs from the first format, however, in that the position of the files in input pathnames is preserved. For example,

```
push lxho9 a/b c a/d/e /tmp/one
```

creates files */tmp/one/b*, */tmp/one/c*, and */tmp/one/e*, while

```
push lxho9 - /tmp/two <<!
a/b
c
a/d/e
!
```

creates files */tmp/two/a/b*, */tmp/two/c*, and */tmp/two/a/d/e*. The second format is useful in combination with *find*(1) to select portions of a directory tree to transfer.

push preserves the file modes and modification times of the files it moves. The original file owner (numeric user ID) is preserved if the effective user ID of the process on the target machine is **root**. Otherwise the files will be owned by the user's login on the target host.

push preserves the name, type and contents of the files it moves with the following exceptions.

Long Names

If the target machine is a SVR4 implementation but the target file system does not support file names greater than 14 characters and the source machine transfers a file with a name greater then 14 characters, the file name will be truncated to 14

characters. The remote side will warn the user for each file name that is truncated.

Long Names

If the target machine is not a SVR4 implementation and the source machine transfers a file with a name greater then 14 characters, the name will usually be truncated to 14 characters. **Note**: A few pre-SVR4 implementations support long file names but the **push** command has no way of knowing if these systems do or do not. The local side will warn the user for each file name that may be truncated.

Symbolic Links

Files of type symbolic link will be preserved unless the '– **L**' option is used or the target machine does not support symbolic links. A transferred symbolic link will be identical to the source file; it may, however, have a different context in the target environment. If the '– **L**' option is used or the target machine does not support symbolic links, the source machine will follow symbolic links with the following behaviors: The symbolic link will be treated as if it were the file type of the file pointed to. If the file does not exist or the user does not have permission to access the file, no transfer will take place and the source machine will issue a warning. If, in following a path that contains a symbolic link that points to a directory, the source machine finds a directory that has already been transferred, it will not transfer it a second time. The source machine, however, will always transfer directories in a path that does not contain a symbolic link even if that directory has already been transferred while following a path that does contain a symbolic link. In both cases the source machine will warn the user for every affected directory.

The **push** and *pull*(1C) commands invoke the **pupu** program (*/opt/dk/bin/pupu*) on the remote system to handle the remote end of all file transfers. If the *srvtab*(4) file on the remote system invokes the **pupu** program with the ´– **r**´ option, file transfers will be restricted to/from the home directory tree on the remote system by disallowing all paths that begin with a leading ´/´ or that contain an embedded ´**..**´ specification. In this restricted mode, only transfers containing PATHs relative to the HOME directory will be accepted.

Multiple Interfaces

If multiple interface boards are installed on the originating host the **push** command will use the default processing to select the interface for the out-going call. See *dkdial*(3X) and *authorize*(1M) for more information.

WARNINGS

If the source host is identical to the target *destination* and the source directory is identical to the target *directory*, then the **push** command will overwrite *filename* and its contents may be destroyed.

The modification times are preserved on transferred files as long as they are earlier in *relative* time (i.e. seconds since the 00:00:00 GMT, January 1, 1970, epoch) than the current relative time on the local machine. If the modification time for a file would date that file in the future on the local machine, the current time on the local machine will be used as the file modification time.

FILES

/opt/dk/bin	directory in which this command resides
/etc/opt/dk/dkhosts	host control file for destination mapping
$HOME/.dkhosts	local host control file for destination mapping

SEE ALSO

pull(1C), *authorize*(1M), *dkdial*(3X), *maphost*(3X), *dkhosts*(4), *srvtab*(4).
time(2) in the *UNIX System V Programmer's Reference Manual*.
find(1) in the *UNIX System V User's Reference Manual*.

NAME

authorize – host authorization service

DESCRIPTION

authorize is the host authorization service. The service establishes the mapping of a valid user ID on the originating host to a valid user ID on the destination host using the *Originating Group* name. This mapping allows a user to bypass the *login*(1) process when connecting to another host on the same AT&T data switch network.

The **authorize** routine maintains the */etc/opt/dk/dkuidtab* file [see *dkuidtab*(4)], which is owned by **root** and contains information on how to map user IDs from incoming calls to valid user IDs. **authorize** also creates and maintains a log file (which you should clean out periodically) of all successful and unsuccessful authorization attempts. The format and name of this log file is documented below.

The **authorize** routine leaves a file in */etc/opt/dk* called *dkuidtab:o* This file is a working copy of the *dkuidtab* file and can be ignored.

The **-l** *login* and **-u** *uidfile* options previously supported with the **authorize** command are now included in the **dkauth** command (refer to the *dkauth*(1C) manual page for details). If a user specifies the **-l** or **-u** option with **authorize** an error message is returned.

Environment Variables

If the shell variable, DKKEY, is set in the user's environment, then that string is used as a matching token when authorizing.

The DKKEY value is also stored in the *dkuidtab*(4) file on the remote *destination*. Thus, from any given originating host and user ID, by changing the value of DKKEY, a user can remotely login as one of a number of different user ID's on a given *destination* host.

Log File

The **authorize** command creates and maintains a log file (owned by **root** with mode **0600**) of all authorization attempts, whether they were successful or unsuccessful. This log is kept in the file */var/opt/dk/log/dkuidlog* and will grow without bound unless it is periodically cleaned by the administrator.

The format of the authorization log file is very similar to the format used by *su*(1). The fields contained in the log are: the string DK, the date and time of the attempted authorization, a *status flag* that indicates the outcome of the attempt, the *Originating Group* of the requester, the numeric user ID of the requester (with a */dkkey_value* appended if there was a DKKEY included in the request), and finally, the login name requested in the attempt.

The *status flag* will be one of four possible values:

- **+** The requester supplied a correct login/password combination and was granted an authorization.

E The requester supplied a correct login/password combination but was denied authorization because the password had expired.

0 The requester supplied a correct login/password combination but was denied authorization because the requested login had a user ID of zero (authorizations may not take place to super-user logins).

– The requester supplied an incorrect login/password combination and was denied authorization.

Here are several sample authorization log entries:

```
DK 10/20 15:50 – nj/jail/hacker 100–guest
DK 10/20 15:50 – nj/jail/hacker 100/dkey–sys
DK 10/20 15:50 0 nj/jail/hacker 0–root
DK 10/20 15:54 – nj/jail/warden 11221–fake
DK 10/20 15:54 – nj/jail/warden 11221–fake
DK 10/20 15:58 E nj/jail/warden 11221–fake
DK 10/20 16:00 + nj/jail/warden 11221–fake
DK 10/20 16:36 + nj/wecare/support 0–guest
```

FILES

/opt/dk/sbin	directory in which this command resides
/etc/opt/dk/dkuidtab	default user ID mapping file
/etc/opt/dk/dkuidtab:o	backup copy of the user ID mapping file
/etc/passwd	password file
/etc/shadow	encoded password file

SEE ALSO

dkauth(1C), *dk*(1C), *dkserver*(1M), *dkdial*(3X), *srvtab*(4), *dkuidtab*(4).
login(1), *mail*(1), *passwd*(1), *su*(1) in the *UNIX System V User's Reference Manual.*

DIAGNOSTICS

When a user attempts to execute the authorization service, several error messages may result. Most of those messages are documented below.

Sorry, whitespace in a DKKEY is forbidden

The DKKEY variable must not contain certain non-printing characters.

The password for this login has expired.
You'll need to get it rejuvenated first.

The **authorize** program will not allow a user to become authorized to a login that has an expired password. The user should "rejuvenate" the password on the remote system and try again.

You're just guessing

This occurs if the user attempts to login incorrectly more than three times in a row.

Sorry, root not allowed

The root user ID cannot be authorized on a remote system.

You are too slow

This is printed when the **authorize** program times out.

Your entry on *destination* **from** *user* **has been deleted.**

This is printed when the `Please login:` prompt is answered with a carriage return. The current authorization (if any) is deleted on the remote *destination* host. *user* is the origination information: *originating group.uid#/DKKEY* of which the DKKEY is optional.

The file *uidfile* **is corrupted, so you can't be authorized.**
Please notify the System Administrator for *host*.

The *uidfile* is the user ID mapping table [see *dkuidtab*(4)]. The **authorize** program will currently detect one type of corruption. If the number of fields in a line of *uidfile* is less than three, this message is printed. The *host* is the remote host specified on the command line.

You are currently being mapped into *user***.**

where *user* is a user name and one of the following self-explanatory messages will be printed:

That user no longer exists on this machine.

But the password has changed.

But the password has expired.

Unsupported option, use dkauth(1C) instead.

The user has specifed the **-l** or **-u** option with **authorize**; these options are no longer supported with the **authorize** command. These options are supported with **dkauth**.

/etc/opt/dk/dkuidtab: No such file or directory.
Call the System Administrator.

This error may indicate that the *dkuidtab* has been deleted.

NAME

dkdaemon – host interface daemon process

SYNOPSIS

dkdaemon [– **i** *interface*] [– **t**] [– **x**] [– **l** *logfile*] [– **v** *verbosity*] [– **a** [*acctfile*]] [– **c** *channels*] [– **b** *urpblocks*] [– **p** *protocol*] [– **w** *windowsize*]

DESCRIPTION

dkdaemon is the host interface daemon process. The primary purposes of **dkdaemon** are to:

1. Retain the *dkux*(7) call processing module on each *dkhs*(7) Stream associated with an AT&T data switch *Common Signaling Channel*.
2. Support accounting, logging and auditing. The auditing records are defined in **dkaudit**(4).
3. Provide dynamic linking and unlinking of *dkhs*(7) Streams for the remote execution [see *dkxqt*(7)] and TLI [see *dktli*(7)] subsystems.

All actions of **dkdaemon** are controlled via command line arguments.

The following arguments select the subsystem or subsystems that will be serviced and may be specified in any combination and order:

– **i** *interface* Service *dkhs*(7) physical interface number *interface*. One – **i** option may be specified for each installed *interface* board.

– **t** Service the *dktm* TLI subsystem driver.

– **x** Service the *dkxqt*(7) remote execution subsystem driver.

When invoked with none of the above options, **dkdaemon** services all possible *dkhs*(7) interfaces and subsystems. When any of these options are used it limits **dkdaemon** to servicing the specified subsystem(s) or interface(s). If multiple **dkdaemon** processes are used with overlapping services, only one **dkdaemon** will support the service. Using the default action may result in the logging of extraneous ERROR or WARNING messages in a less than fully configured system.

Several command line arguments may be used to alter the default logging and auditing modes of **dkdaemon:**

– **l** *logfile* Use *logfile* for activity logging instead of the default file */var/opt/dk/log/dkdaemonlog*. The full pathname for *logfile* must be specified or the *logfile* will be created in */var/opt/dk/log*. This audit trail includes:

A. Activity/status, ERROR and WARNING messages that are described in the DIAGNOSTICS section

B. Connection auditing messages described in *dkaudit*(4)

C. Undocumented debugging messages that appear only at high *verbosity* levels.

– **v** *verbosity* Use *verbosity* as the logging level instead of the default value of **5**. The normal range for log level is **1** through **9**. A larger *verbosity* number means more log detail is desired. Increasing the logging *verbosity* above nine will cause the logging file to grow large with undocumented and useless messages very quickly. The *verbosity* level must be between **1** and **99** (*inclusive*) as shown in the table below.

Log Level	Logged Information	Definition
1	Initialization, file errors, invalid inbound UNIXP messages	**dkdaemon(1M)**
2	Device startups and errors	**dkdaemon(1M)**
4	Server startups and errors	**dkaudit(4)**
5	Outgoing connection requests	**dkaudit(4)**
6	Valid inbound UNIXP messages	**dkaudit(4)**
7	Outbound UNIXP messages	**dkaudit(4)**
8	STREAMS link and unlink	**dkdaemon(1M)**
9	Outbound UNIXP keepalive messages	**dkaudit(4)**
>9	Developer diagnostics	**dkdaemon(1M)**

– **a** [*acctfile*] Enable *dkhs*(7) data transfer, connect time accounting, and optionally use *acctfile* as the accounting file instead of the default file */var/opt/dk/log/dkacct*. The full pathname for *acctfile* must be specified or the *acctfile* will be created in */var/opt/dk/log*. The format of these accounting records is described in *dkacct*(4).

When **dkdaemon** contacts the *dkux*(7) module for each physical *dkhs*(7) interface, it configures several global interface parameters with built-in default values that may be overridden through command line options. These options and the built-in default values are described below. The built-in default values are adequate for all customer installations and these options are provided for advanced host applications and for developer convenience. The parameters apply to all *dkhs*(7) interfaces controlled by the **dkdaemon** process. Use a separate **dkdaemon** process for each *interface* if you wish to vary the parameters for each *interface*.

– **c** *channels* The number of channels to be used by the host interface. Maximum value should be the lower of **512** or the tunable channel limit configured by the *dkhs*(7) driver. The tunable channel limit configured on your machine is defined by the *dkMAXchans* variable in */usr/include/dkit/globals.h*. Minimum value allowable is 3. The built-in default value for *channels* is **64 .**

– **b** *urpblocks* Number of outstanding URP data blocks used for transmit window calculation on *protocol* **4** hosts connected to data switches. The default transmit window for incoming and outgoing calls is divided into *urpblocks* URP data blocks. Applications are free to select their own transmit window parameters [See *dk_info*(3X)] after the call is established, and this option is silently ignored unless the *protocol* is set to **4**.

The value of *urpblocks* must be a number between **1** and **7** (*inclusive*) and the built-in default value is **4**.

– **p** *protocol* Selects the call setup protocol used by the host interface. Two protocol values are supported. The default value of **4** selects the protocol used by newer data switches and is sufficient for most installations. The optional value of **2** selects the protocol used by older data switches. This option does not imply, nor will it provide, support for a direct connection to older data switches. It is provided for some rare instances where older endpoints on the network may be confused by valid parameters of the window size negotiation. In these cases *protocol* **2** may provide a less efficient but always valid window size negotiation. The *dkux*(7) call processing module automatically negotiates down to *protocol* **2** if it is connected to an older data switch. The user, however, must be aware that while this appears to work there are incompatibilities with the older switches that are insurmountable. For the reasons stated above *dkux*(7) will **never** negotiate up to *protocol* **4** if a value of **2** is specified.

– **w** *windowsize* Configures the *Receive Buffer Size* used by the host interface when transmit window negotiation occurs during incoming and outgoing call setup processing. The built-in default value of **1024** should not be changed or loss of data may result. *windowsize* must be a power of **2** in range of **64** and **2048** (*inclusive*).

FILES

/opt/dk/sbin	directory where this command resides
/dev/dk/ctl*X*	*dkhs*(7) *Common Signaling Channel* device for interface *X*
/dev/dknp/daemon	TLI subsystem control device
/dev/dk/dkxmx0	remote execution subsystem control device
/var/opt/dk/log/dkdaemonlog	default log file
/var/opt/dk/log/dkacct	default accounting file

SEE ALSO

dkdiag(1M), *dkitrc*(1M), *dkmaint*(1M), *dkserver*(1M), *dk_info*(3X), *dkacct*(4), *dkaudit*(4), *dksrvlog*(4), *dktli*(7), *dkux*(7), *dkxqt*(7), *dkhs*(7).
chdir(2), *fork*(2), *poll*(2), *signal*(2) in the *UNIX System V System Administrator's Reference Manual*.
ioctl(2), *malloc*(3) in the *UNIX System V Programmer's Reference Manual*.

DIAGNOSTICS

dkdaemon produces status and error messages as appropriate for the circumstances. Option and startup errors are written to standard error, and operational errors and warnings are written to the *logfile* with a timestamp prefix. Connection auditing messages are described in *dkaudit*(4). The value of *Prog* is the name used to start the **dkdaemon** command.

Startup errors include:

Prog: **Illegal Protocol** *Value* **Specified**

The *Value* passed to the – **p** *protocol* command line option was invalid.

Prog: **Illegal Number of Channels** *Value* **Specified**

The *Value* passed to the – **c** *channels* command line option was out of range.

Prog: **Illegal Receive Buffer Size** *Value* **Specified**

The *Value* passed to the – **w** *windowsize* command line option was not a legal power of 2.

Prog: **Illegal Number of Outstanding Blocks** *Value* **Specified**

The *Value* passed to the – **b** *urpblocks* command line option was not within the inclusive range of 1 to 7.

Prog: **Illegal Log Level** *verbosity* **Specified**

The *Value* passed to the – **v** *verbosity* command line option was not within the inclusive range of 1 to 99.

Prog: **Cannot open** "*logfile*" **as log file**

dkdaemon could not open *logfile* as its logging file.

Prog: **Unable to chdir(***Directory***), errno =** *Code*

Error *Code* was returned when **dkdaemon** attempted to *chdir*(2) to *Directory*. Find out why *Directory* does not exist or what is wrong with it.

Prog: **Unable to fork, errno =** *Code*

Error *Code* was returned when **dkdaemon** attempted to *fork*(2) itself into the background.

Status and error messages written to the log file are used to audit events that may be of interest to the administrator and do not necessarily suggest an error. Errors are identified with the word ERROR and tend to be localized to a client subsystem.

acct_start: **ERROR on open(**"*acctfile*"**), errno** *Code*

Error *Code* was returned when **dkdaemon** attempted to open *acctfile* as the accounting file. The process continues with accounting mode disabled.

acct_start: **Accounting Started to** "*acctfile*"

The **dkdaemon** accounting routines successfully opened *acctfile* as the accounting file.

acct_start: **Accounting Disabled**

Accounting mode was shut off either as requested or as a result of a previously logged error. If accounting is desired, **dkdaemon** must be manually restarted.

acct_write: **ERROR write** *Got* **expect** *Length***, errno** *Code*

A write to the accounting file returned *Got* when *Length* was expected. *Code* is significant only when *Got* is – **1.** Any error writing to the accounting file will disable accounting mode.

dkhsstart: **ERROR open(***Device***), errno=***Code*

Error *Code* was returned when **dkdaemon** tried to open *Device* as a *dkhs*(7) controlling channel device.

dkhsstart: **Unit** *interface*: *channels* **Chans, Ver** *protocol*, **Rbuf** *windowsize*, **NurpB** *urpblocks*

Audit of **dkdaemon** attempts to program physical host interface board *interface* with *channels* channels, call processing protocol version *protocol,* per-channel receive buffer size *windowsize* and a default of *urpblocks* URP blocks per transmit window. This message is logged *before* the programming attempt and does not suggest success or failure.

dkhsstart: **ERROR dkhs Unit** *interface* **can't DKDAEMON, errno=***Code*

Error *Code* was returned when **dkdaemon** attempted to start *dkhs*(7) interface *interface* with the DKDAEMON *ioctl*(2). This happens when two **dkdaemon** processes both attempt to start a single interface.

dkhsopen: **ERROR open(***Device***), errno=***Code*

Error *Code* was returned when **dkdaemon** tried to open *Device* in response to an I_LINK request from one of its client subsystems.

dkhsopen: **ERROR DKGETMODCHAN,** *Device* **errno=***Code*

Error *Code* was detected when **dkdaemon** tried to determine the identity of a newly clone-opened *Device* during the service of an I_LINK request from one of its client subsystems.

Prog: **Started, Log Level =** *verbosity*

Written to the log file during **dkdaemon** startup as a record of the session logging verbosity.

Prog: **UNKNOWN SIGNAL** – *Value*

A signal with integer value *Value* was received unexpectedly. See *signal*(2).

Prog: **SIGHUP**

The **dkdaemon** process received a SIGHUP signal that will cause a full re-initialization of all client subsystems.

Prog: **WARNING Termination Requested**

dkdaemon received a SIGTERM request to stop and will honor the request by exiting.

Prog: **Exiting**

This message is logged immediately before **dkdaemon** exits normally.

genstart: ERROR open(*Device***), errno=***Code*

Error *Code* was returned when **dkdaemon** tried to open *Device* as an administrative daemon device.

get_msg: getmsg vdatlen *Length***, ERROR errno =** *Code*

A *getmsg*(2) system call returned error *Code* unexpectedly. A subsystem client request message with up to *Length* bytes of variable length data had been expected.

get_msg: ERROR bad len *Got***, SB** *Length***, ret =** *Return***, flags =** *Flags*

A message of length *Got* was received when **dkdaemon** expected to receive a protocol message of size *Length*. The *Return* code and *Flags* are set by the *getmsg*(2) system call. The contents of the unexpected control and/or data portion of the message are printed, followed by a **service** ERROR that identifies the client driver that sent the improperly formatted message. This message will only be logged if the **dkdaemon** log level is greater than 8.

send_msg: ERROR putmsg type = *Type* **vdatlen** *Length***, errno =** *Code*

Error *Code* was returned when **dkdaemon** attempted to send response type *Type* with variable length data of size *Length*.

parse: ERROR *Name* **Unit** *interface* **bad cmd, type** *Type* **vdatlen** *Length*

An unrecognized command with a type *Type* was received from interface *interface* of subsystem client *Name*. The request included *Length* bytes of variable length data. This is likely caused by a software error in the driver described by *Name*.

parse: ERROR *Name* **Unit** *interface* **vdata missing, type** *Type* **vdatlen** *Length*

A request of type *Type* from subsystem client *Name* unit *interface* did not include the required variable length data buffer. This is likely caused by a software error in the driver described by *Name*.

parse: ERROR *Name***, Unit** *interface* **bad fixed vdata size, type** *Type* **vdatlen** *Got***, SB** *Length*

A request of type *Type* from subsystem client *Name* unit *interface* included a variable length data buffer of *Got* bytes but *Length* bytes were expected. This is likely caused by a software error or a temporary resource exhaustion in the driver described by *Name*.

parse: ERROR *Name* **Unit** *interface* **unexpected vdata type** *Type* **vdatlen** *Length*

A request of type *Type* from subsystem client *Name* unit *interface* included a variable length data buffer when none was expected. This is likely caused by a software error in the driver described by *Name.*

parse: *Name* **Unit** *interface* **Handler type** *Type* **returned** *RetVal*

The Handler for type *Type* client messages returned a failure (with value *RetVal*) while servicing a request for *Name* unit *interface.* This message should be preceded by one or more ERROR messages from the failed Handler.

push_modules: ERROR errno = 255 "dkux"

dkdaemon was unable to push the *dkux*(7) module on a *dkhs*(7) Stream. This error is always followed by a **startstr** message and usually shows that an attempt was made to start two **dkdaemon** processes for a single *dkhs*(7) interface.

push_modules: ERROR errno = *Value* "*Module*"

The more general form of the previous message; **dkdaemon** was unable to I_PUSH the *Module* module on a Stream. Should be followed by additional ERROR messages that identify the Handler, request and client.

the_daemon: ERROR configpoll failed, n= *Integer*

A *malloc*(3) memory allocation failure prevented **dkdaemon** from building tables necessary for its operation and the process has exited. The memory requirements are small and this failure should never happen.

*Prog***: Startup Complete**

All required tables have been built; **dkdaemon** has entered its processing loop and will start trying to bring its clients into service.

the_daemon: WARNING Restart Requested

The **dkdaemon** process has received a restart signal, UNIX System signal '1', and will reinitialize all its client subsystems. This will drop any data switch connections on the client interfaces and restart any server processes on those interfaces.

the_daemon: ERROR Serious Poll errno *Code*

Error *Code* was received unexpectedly from a *poll*(2) system call and the **dkdaemon** process will reinitialize all its client subsystems.

startstr: *Name* **Unit** *interface* **ACTIVE**

Subsystem service client *Name* unit *interface* has been brought into service.

startstr: *Name* **Unit** *interface* **Down, Retrying**

An attempt to bring subsystem client *Name* unit *interface* into service failed and **dkdaemon** will periodically retry until successful or until a fatal error is detected. This may also occur when a **dkdaemon** tries to start an interface that is running diagnostics [see *dkdiag*(1M)].

startstr: *Name* **Unit** *interface* **DROPPED**

An attempt to bring subsystem client *Name* unit *interface* into service failed in a seemingly fatal way and **dkdaemon** has given up trying to service that client. A new **dkdaemon** process may be started for the failed client once the problem has been corrected.

timed_poll: **No Work, Terminating.**

The **dkdaemon** process is exiting because it has DROPPED all its configured clients.

service: **ERROR get_msg** *Name* **Unit** *interface*, Ret= *RetVal*

The **dkdaemon** service manager detected an abnormal return value *RetVal* when trying to read a request from subsystem client *Name* unit *interface.* The client will be restarted.

service: **ERROR parse, halting** *Name* **Unit** *interface*

The request parser and Handler dispatcher returned a fatal status to the service manager for client *Name* unit *interface.* The client will be restarted.

NAME

dkdevs – make special device files for host interfaces

SYNOPSIS

dkdevs [– **i** *interface*] ... [– **c** *channels*] [– **v** *verbosity*]

DESCRIPTION

dkdevs is a program used to make special files in the */dev* directory. **dkdevs** checks for the presence of directories and special files in the */dev* directory required by the host interface software, and makes entries as needed.

dkdevs is called every time the system enters *init* state 2, or it can be run manually by the super-user. When run without the – **i** option, **dkdevs** determines how many host interfaces are installed in the system, and makes the necessary special files. Since **dkdevs** checks for the presence of the required directories and files before creating them, the */dev* directory is not modified if the host interface directories and files already exist.

The following flags are recognized by **dkdevs**:

– **i** *interface* — Specifies the host *interface*s for which the special device files are to be made. Multiple *interfaces* can be specified on one line by specifying multiple '– **i**' options. For example, '– **i0** – **i1**' makes the special device files for *interface*s zero and one. When no '– **i**' option is specified, **dkdevs** examines the UNIX System configuration files to determine how many host *interfaces* are configured on the system, then creates all the appropriate directories and special files.

– **c** *channels* — Specifies the number of raw driver *channels* to create for each *interface*. Maximum value should be the lower of **512** or the tunable channel limit configured by *dkhs*(7) driver. The channel limit configured on your machine is defined by **dkMAXchans** variable in */usr/include/dkit/globals.h*. Minimum allowed value is **3**. The built in default channel value is **64**.

– **v** *verbosity* — Used for debugging and troubleshooting. The level of debugging information can be specified in the range of **1**-**9**. A larger *verbosity* number means more debugging information is desired. If an out of range value is used it will be set to the nearest legitimate value. The **dkdevs** command works silently without the '– **v**' option.

FILES

/opt/dk/sbin — directory where this command resides

/etc/conf/cf.d/mdevice	UNIX Device master file (non-3B2 machines)
/etc/conf/cf.d/sdevice	UNIX Device system file (non-3B2 machines)
/etc/master.d	directory containing UNIX System configuration files (3B2 machines)
/dev/dkpe*X*	special files for physical interface *X* (3B2 machines)
/dev/dk	directory containing "raw" *dkhs(7)* special files
/dev/dkt	directory containing *dkty(7)* terminal driver special files
/dev/dkx	directory containing *dkxqt(7)* remote execution driver special files
/dev/dknp	directory containing **daemon** special file
/dev/dktp*X*	*dktli*(7) transport provider special files for interface *X*

In the */dev/dk** directories, the host interface software will make use of special files in the format *x.yyy* where *x* indicates the interface number, and *yyy* indicates the channel number.

SEE ALSO

dkhs(7), *dktli*(7), *dkty*(7), *dkux*(7), *dkxqt*(7).
master(4), *mdevice*(4), *sdevice*(4) in the *UNIX System V System Administrator's Reference Manual*.

DIAGNOSTICS

The **dkdevs** command reports errors when it cannot open the required UNIX System configuration files, or when it cannot find the host device driver information configured in the system. The command will also report problems encountered making directories and device nodes in the */dev* directory.

NAME

dkipump – download *CommKit* Host Interface module

SYNOPSIS

dkipump *interface* [*download_file*]

DESCRIPTION

dkipump downloads operational software to the *CommKit* interface board specified by the *interface.*

The operational software is taken from *download_file* if specified, or */opt/dk/lib/m.dkit* if no second argument is given. **dkipump** then downloads the specified interface with the download file. After the download is complete, **dkipump** initializes the AT&T data switch interface hardware for operation.

The interface board is downloaded when the *dkdaemon*(1M) is started, but may also be downloaded manually using the **dkipump** command.

FILES

/opt/dk/sbin	directory in which this command resides
/opt/dk/lib/m.dkit	default operational system download file
/dev/dkpe*intf*	download device file

SEE ALSO

dkdaemon(1M), *dkpe*(7).

DIAGNOSTICS

The following error messages may result when invoking **dkipump**.

ksh: /opt/dk/sbin/dkipump: cannot execute

You must be a superuser to execute **dkipump**.

dkipump: Cannot pump interface while dkdeamon is active

The interface must *not* be active when the command is run.

dkipump: Cannot pump - already pumped

The user can only pump the board once.

NAME

dkitrc – rc shell for host interface software

SYNOPSIS

dkitrc { **start** | **stop** }

DESCRIPTION

dkitrc is executed automatically during *init* state transitions with an argument of either **start** or **stop.** Its function is to start/stop the host interface in an appropriate fashion for the prevailing *init* state.

When invoked with an argument of **start,** the **dkitrc** script invokes *dkdevs*(1M) to create appropriate special files and then starts a *dkdaemon*(1M) process to initialize the host interface. Following the startup of the *dkdaemon*(1M) process, **dkitrc** will automatically reserve the channel groups [*dkgroups*(4)] by means of *dkmaint*(1M) and then start a *dkserver*(1M) process.

If invoked with an argument of **stop, dkitrc** stops the *dkdaemon*(1M) and *dkserver*(1M) processes.

FILES

/etc/init.d	directory in which this resides
/etc/rc0.d/K15dkitrc	linked to **/etc/init.d/dkitrc**
/etc/rc1.d/K80dkitrc	linked to **/etc/init.d/dkitrc**
/etc/rc2.d/S15dkitrc	linked to **/etc/init.d/dkitrc**

SEE ALSO

dkdaemon(1M), *dkdevs*(1M), *dkmaint*(1M), *dkserver*(1M), *dkgroups*(4), *dktli*(7).
init(1M) in the *UNIX System V System Administrator's Reference Manual.*

NAME

dkload – load test the host interface and check for errors

SYNOPSIS

dkload *destination*.**dkload** [– **s** *size*] [– **n** *iterations*] [– **l** *logfile*]

DESCRIPTION

dkload is a program used to test whether the host interface to an AT&T data switch is operative and may be used to load test the interface. **dkload** sets up a virtual circuit between two data switch hosts and then sends and receives fixed sized data messages over the circuit. **dkload** must be run as **root.**

dkload may call any host in a data switch network that has the service **dkload** defined in its *srvtab*(4). The host is specified by the *destination* on the command line [see *dkhosts*(4)].

The following flags are recognized by **dkload:**

– **s** *size* The message *size* that will be read and written. The default is **512** and the maximum size is **10240**. A blank must separate the flag from the *size.*

– **n** *iterations* The number of *iterations* that **dkload** should run. The default is zero and is interpreted as a request to run infinitely. A blank must separate the flag from the *iterations.*

– **l** *logfile* File where timing information and error messages are logged. The default *logfile* is */tmp/dkloadlog.*

When running **dkload** between two data switch hosts, the path specified for *logfile* must exist on both hosts. The path on the local machine is relative to the current directory, while the path on the remote machine is relative to the **root** directory (/). A *logfile* will be opened on each host, and each **dkload** process will write to the *logfile* on the host on which it is executing.

When the local and remote host are the same, one *logfile* will be opened, and the **dkload** processes on both sides of the virtual circuit will write to the one file.

The default *logfile* will be used on the remote host if the remote **dkload** cannot open the requested file for writing.

Multiple Interfaces

If multiple interface boards are installed on the originating host **dkload** will use the default processing to select the interface for the out-going call. See *dkauth*(1C) for more information.

FILES

/opt/dk/sbin	directory in which this command resides
/tmp/dkloadlog	default **dkload** log file
/etc/opt/dk/srvtab/dkload	*srvtab*(4) entry
/etc/opt/dk/dkhosts	host control file destination mapping

SEE ALSO

dk(1C), *dkauth*(1C), *dkhosts*(4), *srvtab*(4), *dkhs*(7).
read(2), *write*(2) in the *UNIX System V Programmer's Reference Manual.*

DIAGNOSTICS

The following error and diagnostic messages are reported by **dkload**:

REMOTE could not open *logfile* **as log**

When the remote **dkload** cannot open the requested *logfile*, this error message is written to the default *logfile*.

dkload LOCAL started at *time* **to** *command* **PID=***process_ID*
dkload REMOTE started at *time* **from** *command* **PID=***process_ID*

where *command* is made up from the command line and looks like:

*destination***..:**– **s:***size***:**– **n:***iterations***:**– **l:***logfile*.

This initialization information may be used to verify that the correct arguments were sent and received by **dkload.**

After the first three bytes are transmitted and each time an additional 2048 bytes of data is transmitted, an entry is made in the remote *logfile* specifying the process ID, total number of bytes transmitted, and the time elapsed in seconds. When **dkload** detects bad reads, writes, and corrupted data it writes the error messages to the *logfile*. The messages to the *logfile* are distinguished by the words REMOTE and LOCAL, depending on whether the **dkload** on the remote or local side of the virtual circuit wrote the message.

LOCAL TEST FINISHED OK
REMOTE TEST FINISHED OK

The test finished successfully. The message will also include the process ID, time, and number of messages sent. The *nread* variable indicates the number of bytes read in the last iteration, and *countread* specifies the number of iterations received.

REMOTE TEST FINISHED WITH ERRORS: Data Corruption
LOCAL TEST FINISHED WITH ERRORS: Data Corruption

Some of the characters received did not match the expected characters. Each corrupted character is written to the *logfile* along with the expected character as they are encountered. This message will also include the process ID, time, and number of messages sent. The *nread* variable indicates the number of bytes read in the last iteration, and *countread* specifies the

number of iterations received.

LOCAL WRITE FAILED
REMOTE WRITE FAILED

A *write*(2) unexpectedly failed.

LOCAL READ FAILED
REMOTE READ FAILED

A *read*(2) unexpectedly failed.

LOCAL END OF FILE
REMOTE END OF FILE

The local or remote **dkload** process was killed manually.

LOCAL READ LENGTH ERROR
REMOTE READ LENGTH ERROR

The number of bytes returned from a *read*(2) was incorrect.

LOCAL Read string != Write string
REMOTE Read string != Write string

dkload detected data corruption. Each corrupted byte is written to the *logfile* along with the expected value.

NAME

dkmaint – host interface maintenance

SYNOPSIS

dkmaint – **i** *interface* **[** – **s** |– **r** [– **c** *channel*]] [– **v**]
dkmaint [– **g** |– **C**] *group* [– **v**]

DESCRIPTION

dkmaint is a program used to reset and perform other maintenance operations on host interfaces or channels. When the – **r** option is specified, **dkmaint** sends an *ioctl(2)* to the host interface driver which eventually causes M_HANGUP messages to be issued to each process using the specified channel. Use of the – **i** option and the – **r** option without the – **c** option resets all channels on a given interface. The – **s** option displays the reserved channels for the specified interface. The – **g** option names a channel group. The channels in the group are reserved through the – **g** option, and cleared with the – **C** option. **dkmaint** must be run as **root**.

The following flags are recognized by **dkmaint**:

– **r** Reset-Used to reset the host interface or channel specified. Any active connections on the interface/channel selected will be closed as a result of the reset operation.

– **i** *interface* Specifies which host interface to reset. One and only one *interface* must be specified on the command line. To reset all host interfaces installed on a system, **dkmaint** must be run multiple times.

– **c** *channel* Specifies a channel to reset. Combined with the – **i** option, a single *channel* on a single interface is reset.

– **v** Verbose Option used for debugging and trouble shooting. The **dkmaint** command works silently without the '– **v**' option.

– **s** Requests the display of the interface's reserved channels.

– **g** Specifies the channel group whose channels are to be reserved.

NOTE: Channel 0 or 1 should *never* be a channel group or part of a channel group.

– **C** Specifies the channel group whose reserved channels are to be released.

FILES

/opt/dk/sbin	directory in which this command resides
/dev/dk/ctl*X*	*Common Signaling Channel* device for interface *X*
/etc/opt/dk/dkgroups	channel group control file

SEE ALSO

dkdaemon(1M), *dkgroups*(4), *dkhs*(7), *dkux*(7).
ioctl(2), *close*(2) in the *UNIX System V Programmer's Reference Manual.*

DIAGNOSTICS

The **dkmaint** command reports errors when it cannot open the driver control device for the specified interface. It also reports if the host interface driver returns an error when the *ioctl(2)* request is issued.

WARNINGS

Resetting a channel with **dkmaint** will cause the host interface driver to request that the process currently using the requested channel release [*close(2)*] the channel. This is done by sending an M_HANGUP message from the driver up STREAM to the process. This is designed to look like having the remote side of the connection disappear. The action that the process takes is application dependent.

NAME

dkregister – create *CommKit* Software registration record file

SYNOPSIS

dkregister

DESCRIPTION

dkregister is run by the software installation scripts to create the *CommKit* Software validation record file necessary for using the software. The command can also be run manually to recreate a damaged or incomplete validation record file.

Do not run **dkregister** unless you know or have immediate access to your valid software certificate number and matching software key because the old validation record file will be deleted and the *CommKit* Software will become unusable until the file is recreated.

FILES

/opt/dk/sbin	directory in which this command resides
/etc/opt/.DK_Key_File	validation record file
/etc/opt/.DK_Coord_File	message time coordination file

DIAGNOSTICS

Exits with zero return status on success. Writes a descriptive error message to standard error and exits with a non-zero status if any error is detected.

NAME

dkserver – handle incoming calls from an AT&T data switch network

SYNOPSIS

dkserver [– **i** *interface* **]** **[** – **s** *servername* **]** **[** – **v** *verbosity* **]** **[** – **c** *controltab* **]** **[** – **u** *uidfile* **] [** – **e** **] [** – **S** **] [** – **l** *logfile* **] [** – **C** *channel* **] [** – **r** **]**

dkserver – **t** [– **i** *interface*] [– **s** *servername*]

DESCRIPTION

The **dkserver** program identifies itself to the data switch network as being willing to accept service requests. **dkserver** satisfies those requests by spawning new processes and invoking programs based on the parameters given by the request. The response to each service request is based on how the dialstring and identity of the caller match entries in the *controltab* [see *srvtab*(4)]. Information on how to map the users of remote hosts into their local identities is kept in the *uidfile* [see *dkuidtab*(4)] maintained by the *authorize*(1M) program.

dkserver saves the values of the environment variables at the time it is started and selectively passes them along in the initial environment of all commands invoked. See the description of the – **e** argument and the section on environment variables for more details.

dkserver maintains a log file of all incoming requests in *logfile* which should be cleaned out periodically. You should check this file for lines containing the tokens ERROR or DENIED which will indicate conditions that may need administrative attention.

The default action when no options are specified is to start **dkserver** on the first hardware interface using the machine's *nodename* [see *uname*(2)] with default values for the *logfile*, *verbosity* level, and *controltab*.

The default action of **dkserver** may be altered by the specification of one or more of the following command line arguments:

– **i** *interface* Use *interface* instead of the first hardware interface (interface **0**). The *interface* should be specified as a numeric digit (for example, **1**) If this option is omitted and the environment variable DKINTF is set, **dkserver** will start the interface using the value of DKINTF. If this option is used, the value for DKINTF will be ignored.

– **s** *servername* Use *servername* for the name of the server instead of the *nodename* retrieved from the *uname*(2) system call.

– **v** *verbosity* The *verbosity* parameter controls the amount of information recorded to the *logfile* about each incoming call. The range of values is **1** to **9** (default **6**) where increasing the value results in progressively more log detail. An invalid specification of *verbosity* will be rounded to the nearest legitimate value (that is, values less than **1** will be rounded to **1** and values greater than **9** will be rounded to **9**). See the manual page for *dksrvlog*(4) for

information on the content of the *logfile*.

– **c** *controltab* The directory or file *controltab* should be used as the **dkserver** control table instead of the default directory */etc/opt/dk/srvtab*. The full pathname for *controltab* must be specified. [See *srvtab*(4).]

– **u** *uidfile* The file *uidfile* should be used to map the users of remote hosts into their local identities instead of the default file */etc/opt/dk/dkuidtab* . The full pathname for *uidfile* must be specified. [See *dkuidtab*(4).]

– **e** Expand the base environment variables to include DKINTF and merge them with the current environment. This expanded environment will be passed as the initial environment of all commands invoked. See the section on environment variables for more details.

– **S** The *logfile* is made secure; only readable by **root**. The default permission of *logfile*, as created by the **dkserver**, allows reading by everyone. The *logfile* will only be readable by **root** if this option is specified. However, the **dkserver** will not change the permission of an existing *logfile* if this option is not used.

– **l** *logfile* Use *logfile* instead of the default file */var/opt/dk/log/dksrvlog* . The full pathname for *logfile* must be specified or the *logfile* will be created in the current working directory of **dkserver** (*/etc/opt/dk/srvtab* for multiple *srvtab*(4) files format or "/" for a single *srvtab*(4) file format). [See *dksrvlog*(4).]

– **C** *channel* Specify the logical *channel* number desired to be used as the **dkserver** channel. The channel selected has to be within the channel range of the hunt group configured in the data switch controller database for the *servername* specified through the – **s** option of this command. (For more information about the multiple groups per CPM feature provided by the AT&T data switches – *Datakit* II VCS R3.1.1 and later – read the data switch documentation on the ***enter cpm*** command.) If this option is not used, an arbitrary channel will be selected.

– **r** This option will prevent **dkserver** from running in background mode. This option is not needed if **dkserver** is spawned by *init*(1M).

– **t** Gracefully terminate **dkserver**. No new incoming calls are accepted, and when all existing calls are completed, **dkserver** exits. This ensures proper cleanup of accounting files and should be invoked in system shutdown procedures. The – **i** and – **s** options may be used to specify the interface and **dkserver** name to terminate. If neither – **i** or – **s** is specified, the default **dkserver** (that is, **dkserver** for interface **0** with name *nodename*) will be terminated.

ENVIRONMENT VARIABLES

A base set of environment variables is always passed as the initial environment of all commands invoked by **dkserver.** These variables are always set by the rules given below and will never be overwritten or duplicated by use of the – **e** argument or by environment variables passed as part of a user service request. If the – **e** argument is used, DKINTF will be added to the base set, any other environment variables will be merged with the base set and this expanded environment will be passed to the invoked command. Environment variables that would overwrite an existing variable from the base set will be ignored. If an incoming user service request passes environment variables they will also be merged into the environment variables passed to the invoked command. User requested variables that would overwrite an existing variable from the base set or from the – **e** argument will be ignored.

BASE SET

LOGNAME The LOGNAME environment variable is always passed to the invoked command. It is always set from the value found in the */etc/passwd* file for the user selected using the *user* field of the *srvtab*(4) entry.

HOME The HOME environment variable is always passed to the invoked command. It is always set from the value found in the */etc/passwd* file for the user selected using the *user* field of the *srvtab*(4) entry.

SHELL The SHELL environment variable is always passed to the invoked command. It is always set from the value found in the */etc/passwd* file for the user selected using the *user* field of the *srvtab*(4) entry.

PATH The PATH environment variable is always passed to the invoked command. If PATH is set in the *dkserver* environment, then that value is used. If PATH is not set in the *dkserver* environment, then the default value of */usr/bin:/usr/sbin:/usr/ccs/bin* will be passed to the invoked command.

TZ The TZ environment variable is always passed to the invoked command. If TZ is set in the *dkserver* environment, then that value is used. If TZ is not set in the *dkserver* environment, then the default value of *EST5EDT* will be passed to the invoked command.

DKINTF The DKINTF environment variable becomes part of the base set when the – **e** argument is used. If DKINTF is set in the *dkserver* environment, then that value is used, even if it is set to the null string. If DKINTF is not set in the *dkserver* environment, then it will be set to the match the hardware interface of *dkserver*.

FILES

/opt/dk/sbin	directory in which this command resides
/etc/opt/dk/srvtab	default request control directory
/etc/opt/dk/dkuidtab	default user ID mapping file

/var/opt/dk/log/dksrvlog	default **dkserver** request log file
/etc/passwd	password file
/etc/shadow	encoded password file
/var/adm/utmp	connection accounting
/var/adm/wtmp	connection accounting history
/dev/console	serious error messages
/var/opt/dk/log/dk.*intf***.***srv*	lock file where *intf* is the interface and *srv* is the server name

SEE ALSO

dk(1C), *authorize*(1M), *dkitrc*(1M), *dksrvlog*(4), *srvtab*(4), *dkuidtab*(4).
uname(2) in the *UNIX System V Programmer's Reference Manual*.
init(1M), *pwconv*(1M) in the *UNIX System V System Administrator's Reference Manual*.

CAVEATS

Although it is not recommended, **dkserver** may be spawned by *init*(1M) (instead of the *dkitrc* file) by including an entry in the */etc/inittab* file such as:

```
dksxmpl:3:respawn: /opt/dk/sbin/dkserver -s hostname </dev/null \
 >/dev/null 2>&1
```

If **dkserver** is spawned by *init*(1M):

1. All three standard file descriptors (0, 1, and 2) must be set open and assigned to */dev/null* or the **dkserver** will stall and the server will not come up.
2. The **dkserver** process is not started unless the system has been placed into run level 3 by the **init** command or system reboot.

WARNINGS

If the *controltab* format is changed from a directory format to a flat file format (or vice versa) the **dkserver** must be restarted. **dkserver** must also be restarted after running *pwconv*(1M). For systems that support the password file */etc/shadow*, **dkserver** must also be restarted after the file */etc/shadow* is removed or newly created.

If the PATH is longer than 512 characters, **dkserver** will complain and exit.

NAME

dksrverr - Service Error mailer

SYNOPSIS

dksrverr *loginid service origination* [*optional parameters*]

DESCRIPTION

Unauthorized requests for CommKit Host Interface services will be rejected by the host; **dksrverr** provides a trap mechanism to monitor these invalid attempts. To active this mechanism, you must add the **srvtab**(4) line for **dksrverr** as the last entry in the service's **srvtab**(4) file. (See **srvtab Format** below.)

dksrverr accepts the incoming service request and displays a rejection message (as shown below) to the originator.

```
REQUEST '<service>' from <origination> REJECTED (dksrverr)
```

With a valid *loginid* on the current system, service, origination, and any optional parameters, the following mail message will be sent to the specified *loginid*:

```
Subject: DK Request rejected!

              *** NOTICE ***

Machine '<uname>' has rejected a request for Datakit service!

     Service: <service>
     Originating group ID: <origination>
     Additional info: <optional parameter 1>
     Additional info: <optional parameter 2>
          .
          .
          .
     Additional info: <optional parameter n>

                    DK Service Error Detector (dksrverr)
```

The status of the command is seen in the *dksrvlog* for the server which received the request.

Hex Value	Status
ex0	Mail command was successful
ex100	Incorrect number of arguments
ex200	Invalid *loginid*
ex300	Cannot create pipe for mail
ex400	Could not fork for mail
ex*	Failed mail command exit value. This is returned by

the **mail** command and depends on the high order bit Values may be 8N, 9A-9F. Obtain the correct exit value by and'ing with 0x7F.

srvtab Format

The following shows the format of the **srvtab** line:

```
* service  M  loginid  /opt/dk/sbin/dksrverr dksrverr:loginid:%e:%f.%u:node=%n, \
    module=%m, channel=%c
```

The following parameter descriptions apply:

%c Originating channel number (see also **%m** and **%n**). *channel=%c* is displayed with *node=%n* and *module=%m* on the first line of *Additional info* in the rejection mail message.

%e The requested service from the dialstring.

%f The *Originating Group* name from the dialstring. This has the originating userid, **%u**, as a suffix.

%m The originating module number (see also **%c** and **%n**). *module=%m* is displayed along with *%node=%n* on the first line of *Additional info* in the rejection mail message.

%n The last field starting with *node=%n* is displayed as the first line of *Additional information* in the mail message. (See also **%c** and **%m**.)

%u The user ID of the user placing the call.

Note: Other information can be sent in the mail message by adding additional fields, which must be separated by a colon (**:**), to include other parameters from the dialstring or any specific information.

EXAMPLES

The following example shows the usage of **dksrverr** for the **whoami** service. **srvtab** and **dksrvlog** entries, and mail and rejection messages are similar for the other CommKit Host Interface services.

srvtab entry for **whoami**:

```
# System    Service Flag  User  Program      Initial Parms
#
#
nj/central/* whoami /u  daemon /usr/bin/echo  echo:You are:%u:from:%f...
#
*   whoami     M   root  /opt/dk/sbin/dksrverr  dksrverr:root:%e:%f.%u:node=%n,\
      module=%m, channel=%c:Name of srvtab control directory=%C
```

With the above **srvtab** entry, the following user command would generate the rejection message as shown.

```
dkcu nj/central/jersey1.whoami

REQUEST 'whoami' from nj/south/citya.5035 REJECTED (dksrverr)
```

And, the following mail would be received by the *loginid* responsible for the **dksrverr** command.

```
                        *** NOTICE ***

Machine 'jersey1' has rejected a request for Datakit service!

  Service: whoami
  Originating group ID: nj/south/citya.5035
  Additional info: node=STATE1, module=36, channel=14
  Additional info: Name of srvtab control directory=/etc/opt/dk/srvtab

                            DK Service Error Detector (dksrverr)
```

The associated *dksrvlog* entries for this example are:

```
Apr 27 08:27:01 (17728) [0.253]   REQUEST s=whoami, u=16521=root, r=nj/south/citya
Apr 27 08:27:01 (17728) [0.253]   DEBUG line=5, flag=M, uid=root, file= \
   /opt/dk/sbin/dksrverr, fixedparms=dksrverr:root:%e:%f.%u: node=%n, module=%m, \
   channel=%c:Name of srvtab control directory=%C
Apr 27 08:27:01 (17728) [0.253]   DEBUG linep=/dev/dk/0.253 rlinep=/dev/dk/0.253
Apr 27 08:27:01 (18054) [0.253]   chkidle return 0 file = /opt/dk/sbin/dksrverr
Apr 27 08:27:01 (18054) [0.253]   Send ioctl init = 3
Apr 27 08:27:01 (18054) [0.253]   ORIGIN h=STATE1.36.14, c=/opt/dk/sbin/dksrverr
Apr 27 08:27:01 (18054) [0.253]   SERVICE DENIED:ORIGIN h=STATE1.36.14, \
   c=/opt/dk/sbin/dksrverr
Apr 27 08:27:02 (17728) [0.253]   EXIT l=/dev/dk/0.253 p=18054 e=x0
```

FILES

/etc/opt/dk/srvtab/<**service**>	**srvtab** information
dksrvlog *files*	the specific *dksrvlog* for the server (default */var/opt/dk/log/dksrvlog*)
/bin/mail	to send mail to *loginid*

SEE ALSO

dkserver(1M) *dksrvlog*(4) *srvtab*(4).

CAVEATS

Please note that if **dksrverr** is used for the "-" service, both the Reject message and the mail to *loginid* will show nothing for the service. The **dksrvlog** will show the service request was the "-" service. The reason is that when a user calls in without a service, **dkserver** will default to the "-" service, but the *%e* will not be defined because the user did not request a specific service on the dialstring.

NAME

dkstat – print AT&T data switch interface status information

SYNOPSIS

dkstat [**-i** *interface*] [**-c** *channel* | **-s**] [**-gn**] [**-r**|**x**] [*interval* [*iterations*]]

DESCRIPTION

dkstat retrieves and reports various data switch interface and channel statistics and performance information that is collected by the *dkhs*(7) driver as it operates. This information is formatted and printed on the user's terminal or may be piped into another program for further analysis. Cumulative statistics for an interface may be gathered and reported, or the statistics may be retrieved for a specified interval one or more times.

The available options are:

–**i** *interface* Report statistics for *interface* instead of interface **0** which is the default. Only a single *interface* may be specified per command invocation.

–**c** *channel* Report statistics for a specific single *channel* on the selected interface. The statistics for a channel are only available while that channel is open. Once a channel is closed, the statistics are summed into the channel summary statistics available through the –**s** flag. Only **root** may specify this flag and only a single *channel* may be examined during one command invocation.

–**g** Report global interface statistics for the selected interface.

–**s** Report summary of all channel statistics for the interface. This flag reports the sum of the channel summary statistics and the individual channel statistics for all open channels on the selected interface.

–**n** Suppress all headers. This option allows the output of **dkstat** to be piped into another program for graphing or tabulation of the data.

–**r** Report only **receiver** related statistics.

–**x** Report only **transmitter** related statistics.

interval Gathers and reports statistics for the specified *interval* in seconds. If no *interval* is specified, the cumulative statistics of the type selected are gathered and reported.

iterations Specifies a repeat count for statistics gathering and reporting. The output of **dkstat** is displayed in horizontal format when *iterations* is greater than one. If no *iterations* is specified, the default value is one.

The **Interface Statistics** include statistics about hardware interrupts and errors, driver scheduling and resource problems. The statistics collected are:

x_intr Transmitter interrupts from the host interface board. These interrupts occur whenever the interface board detects the transmit FIFO has made a transition from more than half-full to less than half-full. Transmitter interrupts are also software simulated by *dkhs*(7) whenever a receiver interrupt occurs and the transmit work queue is not empty.

x_xhf The *dkhs*(7) transmitter processing routine detected the transmit FIFO was more than half-full at some point during the transmission of data.

x_hang The *dkhs*(7) transmitter checking routine detected the transmit FIFO was not draining. This usually results in an interface reset. A large **x_hang** count probably indicates a hardware/fiber problem.

r_intr Receiver interrupts from the data switch interface board. These interrupts occur whenever the interface board loads incoming packet data into the receive FIFO causing the FIFO to make a transition from empty to not-empty.

r_nobuf Number of incoming URP data packets dropped by the *dkhs*(7) driver because of buffer exhaustion. This statistic is the count of URP data **packets** dropped and is not a count of user messages lost. A user message may consist of many URP data packets. Dropped packets normally result in the retransmission of data and adversely affect performance by wasting network bandwidth.

r_parity Number of times the fiber optic receiver circuitry reported a parity error for the received data stream. Data with errors is discarded and normally results in a retransmission. A large **r_parity** count probably indicates a hardware problem.

r_frame Number of times the fiber optic receiver circuitry reported a framing error for the received data stream. Data with errors is discarded and normally results in a retransmission. Momentary interruption of the fiber-optic data link often causes a framing error. A large **r_frame** count probably indicates a hardware problem. A bad **clock** or **repeater** module on the data switch node will cause the framing error count to increase rapidly. This will cause communication problems between the node and the host and may result in the loss of service to the host.

r_fbsync The receiver fiber out-of-sync condition detected. This usually results in an interface reset. A large **r_fbsync** count probably indicates a hardware/fiber problem.

r_ovrun Number of times the *dkhs*(7) receiver interrupt routine detected that the hardware receive FIFO was overflowed. A receiver overrun causes data loss and normally results in a retransmission. A large **r_ovrun** count indicates an overloaded system or a problem with higher priority interrupts blocking interrupts to *dkhs*(7).

r_mute The receiver fiber mute condition detected. This usually results in an interface reset. A large **r_mute** count probably indicates a hardware/fiber problem.

open_channels The number of open channels.

The *dkhs*(7) driver collects channel statistics for every active channel. These statistics are available for individual channels while the channels remain active and are available as an interface-wide channel summary whenever the interface is active. Both **Channel Statistics** and **Channel Summary Statistics** contain the following fields:

x_msgs User (STREAMS) data messages transmitted.

x_blocks URP data blocks transmitted. Each user STREAMS message transmitted is broken up into one or more URP data blocks during transmission across the data switch network. The size of the URP data blocks is negotiated at call setup time.

x_bytes User data bytes transmitted.

x_enq Transmitter ENQuires sent. Whenever a transmitted URP data block is not properly acknowledged by the remote endpoint within the expected interval, the URP ENQ character is periodically transmitted until a satisfactory response is received.

x_init Number of URP INIT1 characters transmitted. These initialization characters are transmitted when a circuit is initialized or reinitialized.

r_msgs User (STREAMS) data messages received.

r_blocks Valid URP data blocks received. A complete user message may consist of one or more URP data blocks. The size of the URP data blocks is negotiated at call setup time.

r_bytes User data bytes received.

r_badblk Bad URP data blocks received. Bad URP data blocks result in the transmission of URP REJect characters which cause the bad blocks to be retransmitted. These bad blocks can be received when one or more packets are lost during transmission. This is usually caused by noisy trunks or other network congestion,

but may also be caused by STREAMS message exhaustion.

r_rej URP REJect characters received. The REJ characters are received whenever a transmitted URP data block is lost or damaged during transmission and is rejected by a remote endpoint. The rejected block is retransmitted. URP data blocks are rejected by remote endpoints when one or more packets are lost during transmission. This is usually caused by noisy trunks or other network congestion, but may also be caused by buffer exhaustion at the remote endpoint.

r_enq URP ENQuire characters received. These ENQ characters are transmitted by remote endpoints whenever a transmitted block is not acknowledged within the expected interval.

r_ireq Number of URP INITREQ characters received. URP INITREQ characters are received during circuit initializations or reinitializations.

r_init Number of URP INIT*n* characters received. URP INIT0 and INIT1 characters are received during circuit initializations or reinitializations.

EXAMPLES

dkstat

prints a complete cumulative summary of the statistics associated with interface 0.

dkstat –s 60 60

reports a summary of all channel statistics gathered during 60-second intervals over a period of 1 hour.

dkstat –i1 –c1 120

provides a 2 minute snapshot of channel 1 (*Common Signaling Channel*) activity for interface 1.

FILES

/opt/dk/sbin	directory in which this command resides
/dev/dk/ctl*X*	*Common Signaling Channel* device for interface *X*
/dev/dk/*intf.chan*	individual channel devices

SEE ALSO

dkdevs(1M). *dkhs*(7).

DIAGNOSTICS

Any of the following error conditions may be reported by **dkstat.**

dkstat: **Open Failed, no /dev entry for** *Device*

An attempt was made to open *Device* and no */dev* was found. Create the proper */dev* entry with *dkdevs*(1M) and try again.

dkstat: **Open Failed, device** *Device* **is marked for EXCLUSIVE use**

An attempt was made to open the specified device and the open was refused because *Device* was marked *EXCLUSIVE* by *dkhs*(7).

dkstat: **Open Failed, Interface (***Device***) is down**

dkstat was unable to open the specified device because *dkhs*(7) indicated the interface was down or out of service.

dkstat: **Open** *Device***, errno =** *Code*

The miscellaneous error described by *Code* occurred when **dkstat** attempted to open *Device*.

dkstat: *Ioctl* **errno =** *Code*

The error described by *Code* occurred when **dkstat** attempted to perform the statistics gathering ioctl *Ioctl*.

BUGS

No provision has been made for statistics counters that wrap past zero during a gathering interval.

NAME

dkunlock – create *CommKit* Software validation record file

SYNOPSIS

dkunlock [–**s** *certificate_number* **] [** –**k** *software_key* **]**

DESCRIPTION

dkunlock is run by the software installation scripts to create the *CommKit* Software validation record file necessary for using the software. This command is not intended for use directly and administrators are urged to use the friendlier *dkregister*(1M) command when unlocking their software.

The command line arguments '–**s** *certificate_number*' and '–**k** *software_key*' are optional and **dkunlock** will prompt for the certificate number and/or software key if not provided as an argument. The *certificate_number* and *software_key* are case-insensitive but all characters, including hyphens, must be entered. Do not run **dkunlock** unless you know or have immediate access to your valid software certificate number and matching software key because the old validation record file will be deleted and the *CommKit* Software will become unusable until the file is recreated. This command does not give the administrator a second chance before removing the existing validation record file.

EXAMPLES

To create the validation record file for

certificate number: **FT40-1271941-AF8R**
software key: **A3222-2222-BUEH-VJUA**

execute the following command:

```
dkunlock –s ft40-1271941-af8r –k a3222-2222-bueh-vjua
```

FILES

/opt/dk/sbin	directory in which this command resides
/etc/opt/.DK_Key_File	validation record file
/etc/opt/.DK_Coord_File	message time coordination file

SEE ALSO

dkregister(1M).

DIAGNOSTICS

Exits with zero return status on success. Writes a descriptive error message to standard error and exits with a non-zero status if any error is detected.

NAME

dk_flush – change close processing for an AT&T data switch connection

SYNOPSIS

dk_flush(*fd***,** *flush_time***)**
int fd;
int flush_time;

extern int dk_verbose;

DESCRIPTION

dk_flush changes the close processing for flushing user data queued for transmission on an AT&T data switch connection. The close processing assumes the remote endpoint is actively reading the transmitted data. The argument *flush_time* controls how long the close function will wait for data to drain. *flush_time* can be one of the following:

-1 wait for all of the queued user data to drain to the remote endpoint of the AT&T data switch connection.

0 compute a time delay based on the queued user data to the remote endpoint of the AT&T data switch connection. This assumes data drains at a rate of 10 bytes per second, equivalent to a 110 baud printer.

> **0** wait the specified number of seconds for the queued user data to drain to the remote endpoint of the AT&T data switch connection.

The waiting for the queued user data to drain occurs when the user closes the *fd*, via *close*(2) or *exit(2)*. When the queued data doesn't drain or the remote endpoint disconnects the circuit, *close*(2) fails and returns an error to allow the user to determine the failure condition. This error return only occurs when **dk_flush**(3X) is called. The possible return values from *close*(2) are:

EINTR A signal was received during *close*(2).

ENXIO The remote endpoint of the AT&T data switch connection terminated the connection. The queued data may have been received by the remote endpoint before the connection was terminated. This is dependent on the characteristics of the remote endpoint.

ETIME The requested or computed time expired, and the queued data was not sent to the remote endpoint of the AT&T data switch connection.

In all cases the AT&T data switch connection is disconnected when *close*(2) returns.

FILES

/usr/lib/libdk.so host interface subroutine library

SEE ALSO

dkhs(7), *streamio*(7).

DIAGNOSTICS

A successful return is zero, while an error return is a negative number. **dk_flush** prints a message on *stderr* when it fails, unless the external variable *dk_verbose* has the value of zero(0) before the call to **dk_flush**(3X).

WARNINGS

A successful return from *close*(2) means the remote endpoint has received the data, but doesn't guarantee the user-application has received the data.

NAME

dk_info – get and set information about an AT&T data switch connection

SYNOPSIS

```
#include <dkit/dk.h>

dk_info(fd, cmd, c_infop);
int fd;
int cmd;
dk_intfchan_t *c_infop;

dk_info(fd, cmd, w_infop);
int fd;
int cmd;
dk_urpwin_t *w_infop;

extern int dk_verbose;
```

DESCRIPTION

dk_info is a general-purpose routine retrieving or setting parameters of an open host device. *fd* is an open file descriptor associated with a CommKit stream. *cmd* is one of the following:

DKGETIC Retrieve the host interface number and data switch channel number of the open circuit.

DKGETUW Get the maximum URP block size in bytes and maximum number of outstanding URP blocks. It is valid only for file descriptors associated with *dkty*(7) and *dkhs*(7) streams.

DKSETUW Set the maximum URP block size in bytes and maximum number of outstanding URP blocks. It is valid only for file descriptors associated with *dkty*(7) and *dkhs*(7) streams.

c_infop points to a structure of type *dk_intfchan_t*. This structure includes the following members defined as unsigned short variables:

```
iface;  /* interface board number */
chan;   /* data switch channel number */
```

iface is the interface board number. *chan* is the data switch channel number.

w_infop points to a structure of type *dk_urpwin_t*. This structure includes the following members defined as unsigned short variables:

```
maxblocks;   /* maximum number of blocks */
maxbytes;    /* maximum number of bytes */
```

maxblocks is the maximum number of outstanding URP blocks with the range of values of 1 to 7. *maxbytes* is the maximum number of bytes in an URP block with the range of values of 1 to 65535.

FILES

/usr/lib/libdk.so host interface subroutine library

SEE ALSO

dkdial(3X), *dkerr*(3X).

DIAGNOSTICS

dk_info will return zero for a successful request and a negative number upon error. **dk_info** will print an error message on standard error if the requested action fails unless the external variable *dk_verbose* has been set to 0 before the call is made.

WARNINGS

Setting the window size parameters to values greater than those established at call setup may result in data loss. Data integrity is only guaranteed for window parameters at or below the values established at call setup.

NAME

dk_namer - convert channel number to file name

SYNOPSIS

char *dk_namer(*intf, chan***)**

char *dk_tnamer(*intf, chan***)**

char *dk_xnamer(*intf, chan***)**

int intf;
int chan;

DESCRIPTION

These three routines will convert a host interface and channel number into the UNIX System path name of the device file associated with the raw, tty, or remote execution device.

The raw or pure URP devices are named **/dev/dk/***intf***.***chan***,** the tty processing devices are named **/dev/dkt/***intf***.***chan***,** and the remote execution devices are named **/dev/dkx/***intf***.***chan***,** where *intf* is the interface number and *chan* is the channel number. The names generated by these routines may not exist but are valid for accounting purposes. In general, the devices will exist for all incoming calls but not for outgoing calls. It should also be noted that opening a tty processing device does not guarantee that the tty processing modules will be on the Stream. An outgoing tty device can be selected by the dial routine [see *dkdial*(3X)].

FILES

/usr/lib/libdk.so host interface subroutine library

SEE ALSO

dkdial(3X).

NAME

dk_uxinfo – get and set information about an AT&T data switch connection

SYNOPSIS

#include <**dkit/dk.h**>

dk_uxinfo(*fd, cmd, c_uxinfop***)**
int fd;
int cmd;
dk_uxwin_t * **c_uxinfop;**

extern int dk_verbose;

DESCRIPTION

dk_uxinfo is a general-purpose routine for retrieving and setting parameters of an open host device. This routine is valid only for file descriptors associated with a *dkhs*(7) stream when the *dkux*(7) module is pushed onto that stream. *fd* is an open file descriptor associated with a CommKit stream. *cmd* is one of the following:

DKGET_UXW Get the maximum number of outstanding URP blocks and the log base 2 values of the remote and local URP receive buffers.

DKSET_UXW Set the maximum number of outstanding URP blocks and the log base 2 values of the remote and local URP receive buffers. Values that fall outside of the implementation dependent limits will be adjusted to the closest allowed values. Values of zero will be replaced by the implementation dependent default values. Similarly values that produce unacceptable URP windows will be adjusted to reasonable values. In all cases the actual values used will be returned in the space pointed to by the structure pointer.

c_uxinfop points to a structure of type *dk_uxwin_t*. This structure includes the following members defined as unsigned short variables:

```
nurpblks;       /* number of outstanding URP blocks */
rem_rbuf;       /* log2 value of remote side receive buffer */
loc_rbuf;       /* log2 value of our receive buffer */
```

nurpblks is the maximum number of outstanding URP blocks. This value is used by the local transmitter when it sizes the amount of data that will be transmitted in each URP block. *rem_rbuf* is the log base 2 value of the receive buffer of the remote side. It is used by the local transmitter to limit the maximum number of bytes transmitted in an URP window. *loc_rbuf* is the log base 2 value of the receive buffer of the local side. It is the value advertised in outgoing dialstrings so that a remote transmitter can tailor its URP window.

FILES

/usr/lib/libdk.so host interface subroutine library

SEE ALSO

dkdial(3X), *dkhs*(7), *dkux*(7), *streamio*(7).

DIAGNOSTICS

dk_uxinfo will return zero for a successful request and a negative number upon error. **dk_uxinfo** will print an error message on standard error if the requested action fails unless the external variable *dk_verbose* has been set to zero before the call is made.

WARNINGS

This function provides access to the CommKit drivers for special applications. Its frivolous use will effect the efficiency of data transfer and may even result in data loss.

NAME

dkdial – create an AT&T data switch connection to a remote destination

SYNOPSIS

#include <**dkit/dk.h**>
#include <**dkit/sysexits.h**>

dkitdial (*cmd, dialp***)**
int cmd;
struct dkit_dial ∗**dialp;**

dkdial (*dialstring***)**
char ∗**dialstring;**

dkndial (*dialstring, intf***)**
char ∗**dialstring;**
int intf;

dktdial (*dialstring***)**
char ∗**dialstring;**

dkntdial (*dialstring, intf***)**
char ∗**dialstring;**
int intf;

dkgdial (*dialstring, group***)**
char ∗**dialstring;**
char ∗**group;**

dkgtdial (*dialstring, group***)**
char ∗**dialstring;**
char ∗**group;**

extern int dk_verbose, dk_errno;

DESCRIPTION

dkitdial obtains an available data switch channel and places a call to the destination. It is a general-purpose routine incorporating all of the dialing functions provided by **dkdial, dkndial, dktdial, dkntdial, dkgdial,** and **dkgtdial,** plus additional functionality. *cmd* is the command for **dkitdial.** The value of *cmd* is constructed by or-ing the following flags:

DK_DIAL Do a regular dial. This flag is required.

DK_SELINTF Select an interface board number.

DK_PUSHTTY Push the **dkty**, **ldterm** and **ttcompat** STREAM modules.

DK_USEENV Use the environment variables, DKINTF [interface number(s)], and DKGROUP [channel group(s)], for control information.

DK_SELGRP Use the specified group for connection information.

dialp points to the structure *dkit_dial*. The structure *dkit_dial* includes the following members:

```
char *dest;     /* destination string */
int  intf;      /* interface board number */
char *baud;     /* baud rate string to be added to dialstring */
                /* baud should be set to NULL if there */
                /* is no baud rate string */
char *group;    /* channel group name for outgoing call */
```

dest is the pointer to the destination string. *intf* is the interface board number.

The command flags operate on the following priorities:

DK_SELGRP	overrides DK_SELINTF and DKUSEENV
DK_SELINTF	overrides DK_USEENV
DK_USEENV flags:	
DKGROUP	overrides DKINTF;
DKINTF	if DKINTF is not set, or set to null, then default dialing still occurs
DK_PUSHTTY	when set, DK_PUSHTTY is *always* done, independent of the above variables
baud rate	when the baud rate is non-null, it is used when establishing a connection to the specified destination

When any of the above flags are set in *cmd*, they specify the following:

The DK_SELGRP flag directs that the channel group chosen is taken from *group*. The DK_SELINTF flag directs that the hardware interface number chosen is taken from *intf*. The DK_USEENV flag directs that the hardware interface chosen is set by the environment variable DKINTF or DKGROUP. (See the section below on "Multiple Interfaces" for more information.) If neither DK_SELINTF nor DK_USEENV are set, an interface with an available channel is chosen automatically.

The DK_PUSHTTY flag directs that **dkitdial** automatically push the **dkty**, **ldterm**, and **ttcompat** STREAMS modules in anticipation that the connection will be used for a terminal session. The **dkitdial** call will fail if the **dkty** or **ldterm** STREAMS module cannot be pushed. The call will succeed if the **ttcompat** STREAMS module cannot be pushed, as its functions are generally optional.

baud is the pointer to the baud rate string. The valid baud rate string format is 'BD=' followed by five digits. For example, the string 'BD=09600' is a valid string. *baud* should be set to NULL except for special purpose applications.

dkdial, dkndial, dktdial, dkntdial, dkgdial and **dkgtdial** obtain an available data switch channel, and place a call to the destination specified by *dialstring*. For **dkdial** and **dktdial**, a specific hardware interface is chosen if the environment variables DKINTF or DKGROUP are set. (See the section below on "Multiple Interfaces" for more information.) For **dkgdial** and **dkgtdial** the hardware interface and channel are specified by the **group**. The interface number is taken from *intf* for **dkndial** and **dkntdial**. The **dktdial, dkntdial**, and **dkgtdial** calls automatically push the **dkty, ldterm,** and **ttcompat** STREAMS modules in anticipation that the connection will be used for a terminal session. The routines are otherwise identical.

All calls return the file descriptor of the open circuit when the connection to the destination has been established or a negative number if the connection cannot be made. The **dktdial, dkntdial**, and **dkgtdial** calls fail if the **dkty** or **ldterm** STREAMS module cannot be pushed [see I_PUSH in *streamio*(7)]. The calls will succeed if the **ttcompat** STREAMS module cannot be pushed, as its functions are generally optional.

The circuit is cleared automatically when the last open file descriptor associated with this channel is closed.

MULTIPLE INTERFACES

The *CommKit* Host Interface software supports multiple interface boards, allowing connectivity to multiple data switch networks or redundant connectivity to a single network. The use of multiple interfaces may be controlled by choosing the dial function and its options in conjunction with the environment variables DKINTF and DKGROUP . The following commands: *dk*(1C), *dkcat*(1C), *dkcu*(1C), *dkdo*(1C), *dkload*(1M), *pull*(1C), and *push*(1C) all initiate connections through the network and use the default interface processing to make connections. The default processing and its exceptions are described below.

Default Processing

The default processing will alternate the outgoing connections through every active interface board unless one of the environment variables, DKINTF or DKGROUP is set (i.e., the interfaces are selected in a round robin fashion starting with interface 0 and progressing through the highest configured interface.) If DKINTF is set, then its value will be used as the interface(s) for the outgoing call. If DKGROUP is set, then its value will be used to select the interface(s) and channel(s) for the outgoing call.

DKINTF can be a comma-separated list of interfaces. The dial-out attempts are made in the order defined in the list, until the call is successful or the end of the list is reached. This syntax allows lists in which one interface is preferred over others. For example, '**DKINTF=0,0,1**' will dial over interface 0 twice before dialing over interface 1.

DKGROUP is the name of an entry in the *dkgroups* file.

dkitdial() If the DK_SELINTF flag is set, then the interface number supplied in the **dkit_dial** structure is used. If the DK_SELINTF and DK_USEENV flags are not set, the default processing is used.

dkdial() The default processing is always used.

dkndial() The supplied interface argument is always used, the default processing is never used.

dkgdial() The supplied group is always used.

dkgtdial() The supplied group is always used.

dktdial() The default processing is always used.

dkntdial() The supplied interface argument is always used; the default processing is never used.

Caveats

As stated above, the commands that initiate connections through the network will, unless DKINTF or DKGROUP is set, alternate the outgoing connections through every active interface board. If the interfaces reside in different data switch *Originating Groups*, an authorization [see *authorize*(1M)] made from one interface will not be valid for calls established through one of the other interfaces. Users must explicitly authorize through each interface or must restrict their connections to a single outgoing interface. See the *dkauth*(1C) manual page for more information.

FILES

/usr/lib/libdk.so	host interface subroutine library
/dev/dk/dial	default dialer device
/dev/dk/dial*X*	dialer device for interface *X*
/etc/opt/dk/dkgroups	channel group control file

SEE ALSO

dkauth(1C), *authorize*(1M), *dkmaint*(1M), *dkerr*(3X), *dkgroups*(4), *dkty*(7).
close(2), *open*(2), *getenv*(3), *ldterm*(7), *streamio*(7), *ttcompat*(7) in the *UNIX System V System Administrator's Reference Manual.*

DIAGNOSTICS

All calls return a negative number on an error. Unless the external variable *dk_verbose* had been set to zero, all calls print an error message on standard error when the call fails.

If the failure of these routines is caused by a system call, a value of '**-1**' is returned and the external variable *errno* contains the actual error code. Otherwise, the absolute value of the negative number returned is an error code found in *<dkit/sysexits.h>*.

Further details about the reason for the call failure is stored in the external variable *dk_errno*. See *dkerr*(3X) for additional information regarding *dk_errno*.

BUGS

If the destination is a TY connected device, it is advised to wait one or two seconds after the return of this routine and before writing data to the channel to avoid data loss.

NAME

dkgetepoint, dksetepoint – AT&T data switch endpoint type detection routines

SYNOPSIS

#include <dkit/dk_unixp.h>

int dkgetepoint(*fd, ep_type)*
int fd;
char *ep_type;

int dksetepoint(*fd, ep_type)*
int fd;
char *ep_type;

DESCRIPTION

The host interface subroutine library contains two routines that permit applications to program and retrieve an endpoint type code for incoming circuit connections when using the *dkhs*(7) driver. Homogeneous Datakit II VCS networks supply endpoint type identification in the dialstring allowing *dkserver*(1M) to program the endpoint type with **dksetepoint()** for every incoming call. The endpoint type is set to **DKepUNKNOWN** when connections originate from a Datakit VCS region of the network or when the host is connected to a Datakit VCS node. All endpoint type **#define** constants are kept in the header file */usr/include/dkit/dk_unixp.h*.

dkgetepoint() retrieves the endpoint type code from the circuit associated with *fd* and stores the single character at the location addressed by *ep_type*. The return value is non-zero for success and zero for failure. The value of *ep_type* is undefined on an error return.

dksetepoint() sets the endpoint type code for the circuit associated with *fd* to the single character value found at address *ep_type*. The return value is non-zero for success and zero for failure.

EXAMPLE

The code fragments below illustrate how to retrieve and print the endpoint type code for the circuit associated with *fd:*

```
#include  <dkit/dk_unixp.h>

char      ep_type;
int       ret;

ret = dkgetepoint(fd, &ep_type);
if (ret != 0)
     printf("Endpoint type is '%c'\n", ep_type);
```

FILES

/usr/include/dkit/dk_unixp.h	endpoint type definitions
/usr/lib/libdk.so	host interface subroutine library

SEE ALSO

dkserver(1M).

NAME

dkerr – generate standard AT&T data switch error message

SYNOPSIS

#include <**dkit/dk.h**>

char *dkerr(*error***)**
int error;

DESCRIPTION

dkerr takes an *error* number returned from a call setup attempt and returns a standard error string. The *error* number is stored in the external variable *dk_errno* by all of the *dkdial*(3X) routines. The following is a list of the *error* numbers and their names as defined in <*dkit/dk_unixp.h*>. The header file <*dkit/dk_unixp.h*> is included by the header file <*dkit/dk.h*> .

0 SYS Call Failed

Unknown data switch control computer or remote host error.

1 EdBUSY All channels busy

All assigned ports/channels are in use. All assigned ports/channels are marked as not available by the remote host (this can be caused by the host being down).

2 EdDEADNODE Remote node not answering

A connection cannot be completed because a data switch trunk or data switch control computer somewhere in the network path is inoperable.

3 EdNOANSWER Server not answering

The requested server name is not in service. No interface hardware is assigned to the requested name. The interface hardware assigned to the requested name is not in service or is not operational. The remote server may not answer for reasons of its own. See *dkserver*(1M).

4 EdNONASS Destination not recognized

Some part of the requested name is not defined in the network. The network does not allow this host to have access to the requested name. The requested name is not well formed (too many slashes).

5 EdREORDER All trunk channels busy

One of the data switch control computers has run out of some resource and is unable to process the call at this time.

6 EdEXIST Server already exists

The host attempted to set up a data switch server, but the data switch control computer thinks that there already is a server by that name.

7 EdNOACC Access denied

The call was denied by the remote server. See *dkserver*(1M). This error can also occur if this host attempted to set up a data switch server, but the data switch control computer is not prepared to accept such a setup from this host. This can be caused by: 1) the server name is not defined in the database, 2) the name is not assigned to a group, or 3) the group is not assigned to the CPM-HS module that is connected to this host.

8 EdDIRASS Directory Assistance

The user has requested directory assistance.

10 EdTOOLONG Address too long

The length of the dialstring was larger than the network maximum.

11 EdDIALER Dialer Error

This error has been replaced by errors 22 through 29 below.

12 EdROUTERR Network routing error

The network encountered a routing error.

13 EdTIMOUT Network congestion -- Call timeout. Try again.

The network is congested; the call setup timed out while waiting for a response from some remote entity.

14 EdCONBRK Connection broken. Try again later.

The connection has been broken. This usually happens when some network facility is abruptly removed from service.

15 EdSMGERR Network congestion -- Call initiation failure. Try again.

A call attempt failed due to some lack of resources on the network.

16 EdCHALER Network congestion -- Channel allocation error. Try again later.

A call attempt failed due to a lack of control channel resources on the network.

17 EdCONERR Network congestion -- Connection error. Try again later.

A call attempt failed because the command to set up the virtual circuit failed.

18 EdFWDCER Network congestion -- Call forward error. Try again later.

A call attempt failed due to some lack of resources needed by a trunk to a remote endpoint.

19 EdCOLTRK Network congestion -- Trunk call collision. Try again.

A call attempt failed because the trunk channel picked for the outgoing call was also picked by the remote side for an incoming call. The other side won this one.

20 EdCFGTER Trunk configuration mismatch. Call Network Administrator.

A call attempt failed because the trunk channel picked for the outgoing call is not configured on the trunk on the remote end.

21 EdCFGHER Host configuration mismatch. Call Network Administrator.

A call attempt failed because the host channel picked for the outgoing call is not configured on the network.

The following errors come from the new autodialer.

22 EdDIALNEEDNUM Invalid or missing phone number

The user has neglected to type the phone number required to make a phone call as in **dial.<***valid-phone-number***>**. The phone number is synonymous with the dialstring.

23 EdDIALREORT Call did not go through. Try again later.

The dialer could not complete the call.

24 EdDIALNOIDT No initial dial tone. Try again later.

The autodialer on the called port possibly has a bad telephone line. The port has been marked as bad and removed from the autodialer hunt group.

25 EdDIALNOSDT No secondary dial tone. Try again later.

The autodialer was signaled to wait for a secondary dial tone in the dialing sequence and no such dial tone was detected.

26 EdDIALBUSY Dialed number busy

The call was successfully dialed and a busy signal was detected by the autodialer.

27 EdDIALNOANS No answer from dialed number

The call was dialed successfully and the autodialer didn't get an answer from the dialed number.

28 EdDIALNOCT No carrier tone

The call was dialed successfully and answered; however, no carrier tone was detected by the autodialer.

29 EdDIALBADBAUD Unsupported baud rate

The requested baud rate is not supported by the dialer.

The following errors come from the SIM.

30 EdSIMBUSY SIM all channels busy

All assigned ports/channels on the SIM are in use.

31 EdSIMNOACC SIM no access

The call was denied by the remote side.

32 EdSIMFAIL SIM bad contact

Some failure was encountered while contacting SIM.

33 EdSIMNAK SIM no contact

The SIM rejected the call.

34 EdFACILERR Facility not subscribed

The facility on the SIM was not subscribed to.

35 EdSVCTYPERR Service mismatch

The service on the SIM did not match the request.

36 EdHOSTPROTERR Host protocol error

37 EdXWINBIG Transmit window too big

38 EdRWINSMALL Receive window too small

39 EdCOPENER Open channel error

40 EdURPER Urp error

41 EdCVTVLPER Vlp to dial error

42 EdDSTOOLONG Dialstring too long

43 EdCVTDSER Dial to vlp error

44 EdTRKBUSY Trunk busy

45 EdHOPCNTER Hop count exceeded

46 EdTRKOOS Trunk not answering

47 EdNODIAG No diagnostic channel

48 EdDIFFGOS Mismatched GOS endpoints

The originating and receiving endpoints of a call have the grade of service (GOS) configured differently.

49 EdMOD_NOCIR Insufficient CIR at module

The module receiving the call does not have sufficient resources to support the requested committed information rate (CIR).

The following messages result from a disconnected call.

50 hupEPOINT Endpoint hung up

The endpoint hung up the call. This is not an error.

51 hupNETWORK Network hung up

The network hung up the call.

52 hupSPLICE Splice completed

53 hupSPLICE_FAIL Splice failed

The following errors come from *dkserver*(1M).

130 EdDEVBAD Dkserver: Can't open line. Call System Administrator

The call could not be completed because the host server could not open the host device to make the call.

131 EdMODPUSH Dkserver: Can't push your streams modules

The configured STREAMS modules could not be pushed onto the open channel.

132 EdBADPROTO Dkserver: Invalid protocol requested

The host does not recognize the data switch control computer protocol.

133 EdNOSRVTAB Dkserver: Srvtab not readable. Call System Administrator

The call could not be completed because the host server table (located in */etc/opt/dk/srvtab*) on the remote host was unreadable.

134 EdBADCHROOT Dkserver: Can't chroot. Call System Administrator

The call could not be completed because the host server could not change root to the home directory of the caller on the remote machine.

135 EdOPTION Dkserver: Can't set/get circuit parameters. Call System Administrator

The call could not be completed because the remote *dkserver*(1M) failed to either set the originator's receive buffer size and URP initialization code, or it failed to get the receive buffer size for the receiving channel.

136 EdNOTIDLE Dkserver: Call on a busy device or call collision, try again

The call could not be completed because the remote data switch server detected that the device was not idle. This results from a busy device or from a call collision on the remote host. This is a temporary load related condition.

The following error is returned by the library function.

150 EdBADPARAM Bad parameter

A *dkdial(3X)* routine was called with an invalid parameter.

The following errors come from the old autodialer.

267 Please supply a valid phone number

The user has neglected to type the phone number required to make a phone call as in **dial.<***valid-phone-number***>**. The phone number is synonymous with the dialstring.

523 No response from autodialer. Try again.

The autodialer on the called port failed to respond. The port has been marked as bad and removed from the autodialer hunt group.

779 Auto dialer failed to initiate call. Try again.

The autodialer on the called port responded but failed during dialing. If this message appears more than twice in succession, contact the data switch Network Administrator.

1035 No initial dial tone detected

The autodialer on the called port possibly has a bad telephone line. The port has been marked as bad and removed from the autodialer hunt group.

1291 No secondary dial tone detected

The autodialer was signaled to wait for a secondary dial tone in the dialing sequence and no such dial tone was detected.

1547 Dialed number is busy

The call was successfully dialed and a busy signal was detected by the autodialer.

1803 No answer from dialed number

The call was dialed successfully and the autodialer didn't get an answer from the dialed number.

2059 No carrier tone was detected

The call was dialed successfully and answered, however, no carrier tone was detected by the autodialer.

2315 Could not complete your call. Try again.

The call could not be completed for the following possible reasons: 1) the autodialer failed to complete the dialing sequence, or 2) the port connected to the autodialer was removed from service during the call.

2571 Unsupported baud rate

The requested baud rate is not supported by the dialer.

FILES

/usr/lib/libdk.so host interface subroutine library

SEE ALSO

dkserver(1M), *dkdial*(3X), *srvtab*(4).
ioctl(2) in the *UNIX System V Programmer's Reference Manual.*

NAME

dkget_goslevel, dkset_goslevel – get and set transmit GOS level for an AT&T data switch connection

SYNOPSIS

#include <**dkit/dk.h**>
#include <**dkit/rdfp.h**>

int dkget_goslevel(*fd, goslevel)*
int fd;
int *goslevel;

int dkset_goslevel(*fd, goslevel)*
int fd;
int *goslevel;

DESCRIPTION

The host interface subroutine library contains two routines that permit applications to modify the transmit GOS level for circuit connections when using the *dkhs*(7) driver. The default transmission mode is GOS5 which provides flow control and error correction via retransmission. GOS4 provides flow control, but no error correction. GOS3 does not provide either flow control or error correction. All GOS changes were made in accordance with the *Universal Receiver Protocol (URP) Internal Interface Specification* Issue 2, December 1991 (700-283). The transmission mode should ONLY be modified when the effect on both the application and network is fully known.

dkget_goslevel() retrieves the transmit GOS level from the circuit associated with *fd* and stores the integer value at the location addressed by *goslevel*. The return value is non-zero for success and zero for failure. The value of *goslevel* is undefined on an error return.

dkset_goslevel() sets the transmit GOS level for the circuit associated with *fd* to the integer value found at address *goslevel*. The goslevel value must be one of the following: GOS3, GOS4, or GOS5, as defined in */usr/include/dkit/dk.h*. The return value is non-zero for success and zero for failure.

EXAMPLE

The code fragment below illustrates how to retrieve and print the GOS level for the circuit associated with *fd:*

```
#include  <dkit/dk.h>
#include  <dkit/rdfp.h>

int       goslevel;
int       ret;

ret = dkget_goslevel(fd, &goslevel);
if (ret != 0)
     printf("GOS level is '%d'\n", goslevel);
```

FILES

/usr/include/dkit/dk.h	supported transmit GOS level definitions
/usr/lib/libdk.so	host interface subroutine library

SEE ALSO

Universal Receiver Protocol (URP) Internal Interface Specification Issue 2, December 1991.

NAME

dkleveld, dkeof, dkbreak, dkusb, isdkleveld, isdkeof, isdkclosed – data switch Level–D and unsequenced data support routines

SYNOPSIS

#include <**sys/types.h**>
#include <**dkit/rdfp.h**>
#include <**dkit/dk_urp.h**>

int dkleveld(*fd, msg, len***)**
int fd;
dk_lvld_t *msg;
int len;

int dkeof(*fd***)**
int fd;

int dkbreak(*fd***)**
int fd;

int dkusb(*fd, byte1, byte2***)**
int fd;
uchar_t byte1;
uchar_t byte2;

int isdkleveld(*fd***)**
int fd;

int isdkeof(*fd***)**
int fd;

int isdkclosed(*fd***)**
int fd;

DESCRIPTION

The host interface subroutine library contains a number of routines to support the use of AT&T data switch URP Level–D control codes and the transmission of unsequenced data. These routines are used internally by the host interface administrative and applications programs.

dkleveld transmits a *msg* of arbitrary *len* containing mixed URP Level–D control codes and data on the circuit associated with *fd*. The mixed control and data message is scheduled for transmission after any data already queued for the circuit. *msg* should point to an array of **dk_lvld_t** objects containing the mixed control and data, and *len* is expected to be the length of the array in bytes. Each data item in the array must be made with the *dkDmkdata()* macro and each control code item must be made with *dkDmkctl()*. The length of *msg* may not exceed the system limit for control messages. A non-zero value is returned on successful completion.

dkeof transmits a URP Level–D `EOF` control code on the circuit associated with *fd* by making an internal call to **dkleveld**. The `EOF` control code is scheduled for transmission in a separate URP block after any data already queued for the circuit. A non-zero value is returned on successful completion.

dkbreak transmits a URP Level–D `BREAK` control code on the circuit associated with *fd* by making an internal call to **dkleveld**. The `BREAK` control code is scheduled for transmission in a separate URP block after any data already queued for the circuit. A non-zero value is returned on successful completion.

dkusb transmits two bytes of unsequenced data on the circuit associated with *fd*. The unsequenced data is scheduled for the next transmission opportunity on the circuit, bypassing any data already queued for the circuit. A non-zero value is returned on successful completion.

isdkleveld tests whether the top message at the Stream-head associated with *fd* contains a URP Level–D control code message. If an M_PROTO message containing a URP Level–D control code is found, the control code is returned. Zero is returned if there is no message at the Stream-head or if the message does not contain a URP Level–D code.

isdkeof tests whether the top message at the Stream-head associated with *fd* contains a URP Level–D control code message with an `EOF` code value. If an M_PROTO message containing a URP Level–D control code is found and the control code is an `EOF` code, a non-zero value is returned. Zero is returned if there is no message at the Stream-head or if the message does not contain a URP Level–D `EOF` code.

isdkclosed tests whether the circuit associated with *fd* has been disconnected. A non-zero value is returned if the circuit has been disconnected and a zero value is returned if the circuit appears to be still active.

EXAMPLE

The code fragments below illustrate how to use the **dkleveld** routine to transmit the string 'abc' sandwiched between URP Level-D `BREAK` control codes on the circuit associated with *fd*:

```
#include  <sys/types.h>
#include  <dkit/rdfp.h>
#include  <dkit/dk_urp.h>

dk_lvld_t msg[] = {
          dkDmkctl(URPdBREAK),
          dkDmkdata('a'),
          dkDmkdata('b'),
          dkDmkdata('c'),
          dkDmkctl(URPdBREAK)
};

ret = dkleveld(fd, msg, sizeof(msg));
```

FILES

/usr/lib/libdk.so host interface subroutine library

SEE ALSO

dkhs(7), *streamio*(7).

WARNINGS

If a bad Level-D control code is specified, no error will be returned but subsequent use of the file descriptor may result in errors. This will happen because an M_ERROR is returned which makes all system calls, except *close()* and *poll()*, FAIL.

NAME

dkmgr – establish an AT&T data switch server

SYNOPSIS

```
#include <stdio.h>
#include <dkit/dk.h>
#include <dkit/dkmgr.h>

struct mgrmsg *dkmgr(srvname)
char * srvname;

struct mgrmsg *dknmgr(srvname, intf)
char * srvname;
int intf;

struct mgrmsg *dknnmgr(srvname, intf, srvchan)
char * srvname;
int intf, srvchan;

void dkmgrack(chan, windowsz)
int chan;
short windowsz;

void dkmgrnak(chan, error, windowsz)
int chan, error;
short windowsz;

int dkinit(fd, mode)
int fd, mode;

int dkcktcfg(srvfd, chan, windowsize, protocol)
int srvfd, chan;
short windowsize, protocol;

int dk_chk_idle(intf, chan)
int intf, chan;

int dk_reset_ckt(intf, chan)
int intf, chan;

extern FILE   *logf;
extern int    loglvl;
extern short  recv_win;
extern short  nurpblks;
extern short  dk_protocol;
extern int    dksrvfd;
extern int    dksrvfatal;
extern struct stat *ctl_stat;
```

DESCRIPTION

This set of functions and external variables may be used to establish services on an AT&T Data Switch, receive call requests, initialize and configure circuits, and accept or reject the call request. A program that calls these functions requires some initialization as well as knowledge of the external variables. See the section **EXTERN VARIABLES** below.

dkmgr, **dknmgr** and **dknnmgr** set up the calling process as a data switch server named *srvname* and receive call setup requests. The hardware interface chosen is zero for **dkmgr** and *intf* for **dknmgr** and **dknnmgr**. The hardware channel chosen is the lowest free channel for **dkmgr** and **dknmgr** and *srvchan* for **dknnmgr**. These routines return a pointer to an incoming service request of the following format:

```
struct mgrmsg {
 short          m_chan;       /* channel number of connection       */
 unsigned short m_tstamp;     /* time stamp of request              */
 short          m_urp;        /* how to initialize URP              */
 short          m_windowsize; /* log value of call originator's
                               * receive buffer size                */
 char *         m_protocol;   /* protocol options from user         */
 char *         m_origtype;   /* type of originating device         */
 char *         m_parm;       /* parameter string from user         */
 char *         m_uid;        /* param from system/user, aka UID    */
 char *         m_dial;       /* dial string entered                */
 char *         m_source;     /* originator as known to remote node */
 char *         m_lname;      /* originator as known to local node  */
 char *         m_service;    /* service type requested by user     */
 char *         m_baudrate;   /* baud rate of originator            */
 char *         m_lflag;      /* L == call from local node,
                               * R == call from a remote one.       */
 char *         m_srcnode;    /* originating node (last segment)    */
 char *         m_srcmod;     /* originating mod                    */
 char *         m_srcchan;    /* originating channel                */
 char *         m_cflag;      /* call flag: F=first, P=previous     */
 char *         m_errmsg;     /* possible error msg if m_chan <= 0  */
 char *         m_modtype;    /* mod. type: field one of fifth line */
} ;
```

Field *m_chan* gives the channel number assigned to the incoming request. This channel number is associated with the CommKit device name returned by **dk_namer**(3X). The data switch circuit is already established but the device needs to be opened, initialized and configured. Field *m_tstamp* gives the time stamp of the request. Field *m_urp* indicates how the host interface should initialize URP. Field *m_windowsize* is the log value of the call originator's receive buffer size. Field *m_protocol* gives the protocol options from the user dialstring. Field *m_origtype* is reserved for a subparameter of the protocol field. Field *m_parm* contains the parameters from the user dialstring. String *m_uid* is a sequence of characters extracted from the call setup request message. It may contain optional subfields separated by "." or "/" characters. This field may represent an octal user ID

(if the first character is zero), a decimal user ID (if entirely numeric), or a character login.

The string *m_dial* gives the dialstring used by the calling process. The string *m_source* is the group name (area/exch/group) of the port in the network where the call originated. String *m_lname* is the trunk, (data switch group name), as known to the local node. The requested service appears in *m_service*. Fields *m_srcnode*, *m_srcmod*, and *m_srcchan* are the originating node, module and channel, respectively. Fields *m_srcmod* and *m_srcchan* are in decimal.

Field *m_cflag* is the call flag indicating whether this is the first call from this host or if there have been previous calls from this host. String *m_errmsg* gives a text error message which is valid only if the returned *m_chan* is less than or equal to zero. Field *m_modtype* gives the module type information which is intended to be used internally by the network.

The server program must set the URP initialization mode for the requested channel by calling **dkinit**. The **dkinit** function is passed two parameters: *fd* is the open file descriptor of the device and *mode* the URP initialization instructions. *mode* should be the *m_urp* value returned in the mgrmsg structure but may be any combination of the bitwise OR of zero, *DKXINIT* or *DKRINIT*. **dkinit** returns a non-negative value on success. Otherwise, a value of -1 is returned and the external variable *errno* is set to indicate the error.

The server program must also configure the device via a call to **dkcktcfg**. This function sets the device to the active state and passes the URP window size to the transmitter. This function uses the open file descriptor *srvfd* of the server channel, found in the external variable *dksrvfd*. The channel *chan* and window size *windowsize* are taken from call request structure members *m_chan* and *m_windowsize* respectively. The *protocol* is the protocol value from the *dk_protocol* external variable. **dkcktfg** returns a non-negative value on success. Otherwise, a value of -1 is returned and the external variable *errno* is set to indicate the error.

The server program is also expected to call either **dkmgrack** or **dkmgrnak** to complete the call. The first acknowledges and accepts the call, the second rejects the call. The first argument for both is *chan* the channel number from the call request. The second argument to **dkmgrack** is *windowsz*, it is the window size that is returned to the calling endpoint. Its value is found in the external variable *recv_win*. The second argument to **dkmgrnak** is the error code giving the reason the call was rejected. The third argument is the window size that is returned to the calling endpoint. Its value is found in the external variable *recv_win*.

The server program may also make use of the **dk_chk_idle** function to see if the device specified by *intf* and *chan* is idle and not in use. This check is made via the common signaling channel and returns zero (EX_OK) if the channel is idle and EX_UNAVAILABLE if it is not. For obvious reasons, the program should not have the requested channel open when this check is made. If the circuit is not idle the server can attempt to force it into the idle state by calling **dk_reset_ckt** to reset the device, specifying *intf* and *chan*. This function also returns zero (EX_OK) if the request was accepted and EX_UNAVAILABLE if it is not. Both functions will return EX_OSFILE or EX_NOPERM if there are problems opening or accessing the

common signaling channel. There is no guarantee that the circuit will be returned to the idle state, but it will be rendered unusable until any processes that have the device open, *close*(2) it.

EXAMPLE

The code fragments below illustrate how to use dkmgr(3X) routines to create a server, receive and accept incoming calls.

```
#include   <stdio.h>
#include   <dkit/dk.h>
#include   <dkit/dkmgr.h>

char       cktname[], srvname[];
int        child, errcode, fd, infc, retval;
struct mgrmsg    *inmsgp;

extern short     dk_protocol, recv_win;
      .
      .
      .
while(1) {
      inmsgp = dknmgr(srvname, infc);
            if ( inmsgp->m_chan <= 0 ) {
                  error handling;
                  continue;
      }
            /* Got an incoming call - validate it */
            retval = validate(inmsgp);
            if ( retval < 0 ) {
                  error handling;
                  continue;
            }
            child = fork();
            if ( child==0 )
                  /* Child process breaks out of while() loop */
                  break;
            /* Parent stays in while loop to wait for incoming calls */
}

/* This code is executed by the child process */

/* Get the dk circuit device name */
cktname = dk_namer(infc, inmsgp->m_chan);

/*
Check that dk circuit is idle before attempting open.
This call will fail if not executing as 'root'.
*/
retval = dk_chk_idle(infc, inmsgp->m_chan);
```

```
if ( retval != EX_OK ) {
      /* You can either do error handling OR ... */
      errcode = <value indicative of this error>;
      dkmgrnak(inmsgp->m_chan,errcode,recv_win);
      error handling;
      exit;

      /* ... try to reset the dk circuit */
      retval = dk_reset_ckt(infc,inmsgp->m_chan);
      if ( retval != EX_OK ) {
            errcode = <value indicative of this error>;
            dkmgrnak(inmsgp->m_chan,errcode,recv_win);
            error handling;
            exit;
      }
      /*
      Check that dk circuit is idle after reset
      */
      for ( i=0; i < TOO_MANY; ++i ){ /* programmer defines TOO_MANY */
            if ( dk_chk_idle(infc, inmsgp->m_chan) == EX_OK )
                  break;
            sleep(1);  /* delay before checking again */
      }
      if ( i==TOO_MANY ) {
            /* The channel is still not idle */
            errcode = <value indicative of this error>;
            dkmgrnak(inmsgp->m_chan,errcode,recv_win);
            error handling;
            exit;
      }
}

/*
If we got here, the dk circuit should be available:
either because it was idle or because it was reset.
*/
fd = open(cktname, O_RDWR);
if ( fd < 0 ) {
      errcode = <value indicative of this error>;
      dkmgrnak(inmsgp->m_chan,errcode,recv_win);
            error handling;
            exit;
}
/* dk circuit has been opened and can be initialized */
retval = dkinit(fd, inmsgp->m_urp);
if ( retval < 0 ) {
      close(fd);
      errcode = <value indicative of this error>;
      dkmgrnak(inmsgp->m_chan,errcode,recv_win);
```

```
      error handling;
      exit;
}

retval = dkcktcfg(dksrvfd, inmsgp->m_chan,
      inmsgp->m_windowsize, dk_protocol);
if ( retval < 0 ) {
      close(fd);
      errcode = <value indicative of this error>;
      dkmgrnak(inmsgp->m_chan,errcode,recv_win);
      error handling;
      exit;
}

dkmgrack(inmsgp->m_chan, recv_win);
/* dk circuit is ready for use; you can now read(fd) and write(fd). */
exec(application);
         .
         .
         .
```

EXTERN VARIABLES

Several external variables are set by calling **dkmgr**, **dknmgr** or **dknnmgr** and should not be changed. These variables are:

dksrvfd – the file descriptor of the server channel.

recv_win – the receiver window size for this host.

nurpblks – the number of allowable outstanding URP blocks for this host.

dk_protocol – the call set-up protocol of the AT&T data dwitch.

dksrvfatal – an indication that the requested service name was rejected by the data switch because it is already running or because the name is unknown or out of service.

One external variable is initialized to a default value and may be changed as needed.

loglvl – the log level of the messages written to the log file. The initial value of the variable is 6.

There are two external variables that must be set by the calling program or they will not be used. They are:

logf – the return value of an **fopen(3)** library call of the file that will be used as the log file.

ctl_stat – a pointer to a **stat** structure that was passed to the **stat**(2) system call for the common channel device. The interface number of the common channel device must match the interface number of the server.

FILES

/usr/lib/libdk.so	host interface subroutine library
/dev/dk/dial	default dialer device
/dev/dk/dial*X*	dialer device for interface *X*
/dev/dk/ctl*X*	*Common Signaling Channel* device for interface *X*

SEE ALSO

dkdial(3X), *dk_namer*(3X).
stat(2) and *fopen*(3C) in the *AT&T UNIX System V Programmer's Reference Manual.*

DIAGNOSTICS

In case of a call setup error or a data switch controller restart, a message is formatted, a pointer to the message is placed in *m_errmsg*, *m_chan* is set to be less than or equal to zero, and a normal return is done.

NAME

dksplice – splice two AT&T data switch connections together

SYNOPSIS

#include <**dkit/sysexits.h**>

int dksplice(*fdout, fdin***);**
int fdout;
int fdin;

DESCRIPTION

dksplice arranges for two data switch circuits to be connected together. The splicing process must have a valid file descriptor for each circuit and both circuits must be on the same host interface. The two file descriptors *fdout* and *fdin* are passed as arguments to **dksplice**. Normally, *fdout* corresponds to an outgoing call made earlier using *dkdial*(3X); while *fdin* corresponds to an incoming tty call from a data switch. However, any two data switch channels owned by the process may be spliced. Extreme care should be used under these circumstances and the user is responsible for ensuring that the endpoints of the splice are compatible and that a termination protocol is agreed on.

After the splice takes place, the system automatically arranges for the new data switch circuit to initialize its URP receivers and transmitters.

When the splice successfully completes, **dksplice** closes both *fdin* and *fdout* before returning. Since these circuits are now unusable, the user must also close any *dup*(2) copies of them. For example, in the case where one of the spliced circuits was used for login, standard output and standard error must be closed since they are duplicated from standard input.

FILES

/usr/lib/libdk.so host interface subroutine library

SEE ALSO

dkdaemon(1M), *dkdial*(3X), *dksplwait*(3X), *dkhs*(7), *dkux*(7).
dup(2), *ioctl*(2) in the *AT&T UNIX System V Programmer's Reference Manual*.

DIAGNOSTICS

When **dksplice** fails, it returns a negative value to the user and sets *errno,* but it does *not* close the file descriptors *fdin* or *fdout*. Any STREAMS modules that were pushed on the Stream before the call to **dksplice** will no longer be on the Stream after **dksplice** returns. Hence, if a user wishes to use the two circuits after **dksplice** has failed, the user must reconfigure the circuits.

If you are expecting to recover from failed **dksplice** calls, it is recommended that you save your terminal settings prior to the **dksplice** call. Upon return from a failed splice, you can then push the required modules (**dkty, ldterm, ttcompat**, etc.) and reset your terminal settings.

The return codes and [*errno*] settings are described below.

EX_OSFILE: **dksplice** was unable to push *dkux*(7) onto one of the Streams associated with *fdin* or *fdout*.

EX_DATAERR: **dksplice** does not recognize one or both of *fdin* or *fdout* as a Stream associated with a data switch circuit. This error can also occur when multiple host interfaces are installed, and *fdin* and *fdout* are not associated with the same physical interface.

EX_UNAVAILABLE: The DKSPLICEPREP *ioctl*(2) failed for one of the following reasons:

- [EINVAL] The data structure associated with the DKSPLICEPREP *ioctl*(2) was not found or was not of the proper size, the channels specified in the DKSPLICEPREP *ioctl*(2) data structure are not owned by the user, or the user asked to splice two file descriptors which correspond to the same channel.
- [ENOENT] The *Common Signaling Channel* has not been configured by the daemon process, *dkdaemon*(1M), and thus the interface is down.
- [ENODEV] One or both of the channels to be spliced are not open circuits.
- [EIO] A software error of unknown cause has occurred.
- [ENOSPC] The *dkux*(7) STREAMS module was unable to set up the splice request because of lack of buffers.

EX_SOFTWARE: The DKSPLICE *ioctl*(2) failed for one of the following reasons:

- [EINVAL] The data structure associated with the DKSPLICE *ioctl*(2) was not found or was not of the proper size. This *errno* value also occurs when the user asked to splice two file descriptors which correspond to the same channel.
- [ENOENT] The *Common Signaling Channel* has not been configured by the daemon process, *dkdaemon*(1M), and thus the interface is down.

[ENODEV]	One or both of the channels to be spliced are not open circuits.
[EIO]	A software error of unknown cause has occurred.
[ENOSPC]	The *dkux*(7) STREAMS module was unable to set up the splice request because of lack of buffers.
[ENOMEM]	Circuits involved in the *ioctl*(2) call are not in the proper state.

NAME

dksplwait – wait for an AT&T data switch circuit to be reconnected

SYNOPSIS

int dksplwait(*fd***);**
int fd;

extern int dk_splwtime;

DESCRIPTION

dksplwait allows the target of a splice to wait for the splice to complete. The *fd* argument indicates the circuit to be waited on.

Normally the following actions are taken to maintain synchronization in a splicing scenario:

CALL ORIGINATOR	*SPLICING HOST*	*TARGET HOST*
Makes call to splice application.		
	Makes call to target via *dkdial*(3x). Waits on read.	
		Writes to splicing host. Waits for splice to complete with **dksplwait**.
	Splices circuits with *dksplice*(3X). Exits.	
Continues conversation with target.		Continues conversation with call originator.

dksplwait will return immediately if the reconnection has already taken place.

The parameter, *dk_splwtime*, if set to a positive integer by the user, causes **dksplwait** to time out after the specified number of seconds. Setting *dk_splwtime* to **-1** will cause **dksplwait** to wait indefinitely for the splice to complete. By default, *dk_splwtime* is set to **-1**. It is recommended that *dk_splwtime* be used to guarantee recovery from the error case where, for some reason, the splice does not take place.

FILES

/usr/lib/libdk.so host interface subroutine library

SEE ALSO

dksplice(3X), *dkhs*(7), *dkux*(7).

DIAGNOSTICS

On successful completion, a value of zero is returned. Otherwise, a value of **-1** is returned and *errno* is set to indicate the error. An *errno* value of `EINVAL` indicates that *dk_splwtime* has been set to a value less than **-1**. An *errno* value of `ENOENT` means that the interface is not available. An *errno* value of `ETIME` indicates that **dksplwait** timed out before the splice completed.

NAME

dktr_splice – transparent splice facility

SYNOPSIS

#include <**dkit/dk.h**>
#include <**dkit/dkmgr.h**>

dktr_splice(*mgrp, dialstring, intf, uid***)**
struct mgrmsg *mgrp;
char * dialstring;
int intf;
uid_t uid;

dktr_osplice(*mgrp, dialstring, intf, schan, nchan, uid***)**
struct mgrmsg *mgrp;
char * dialstring;
int intf, schan, nchan;
uid_t uid;

dktr_call(*dialstring, intf, schan, nchan, baud, rbuf, uid***)**
char * dialstring, * baud;
int intf, schan, nchan, rbuf;
uid_t uid;

extern FILE * logf;
extern int loglvl;

DESCRIPTION

This set of functions is provided for the use of security servers that wish to make use of the transparent splice facility available on newer releases of the AT&T Data Switch. **dktr_splice** and **dktr_osplice** will request a transparent splice for an incoming call request represented by *mgrp* and the destination represented by *dialstring.* The interface of the outgoing call is given in *intf* and must be the interface of the incoming call request. The outgoing call will be placed as user *uid*, assuming *uid* is valid on the splicing host. **dktr_osplice** has two additional parameters that attempt to place the outgoing call on channel in the range of *schan* to (*schan* plus *nchan*). These two parameters allow the outgoing calls to be confined to some known originating group on the data switch.

Both of these function will place the outgoing call using the baud rate and receive buffer size from the incoming call request and then will answer the incoming call request with the receive buffer size from the answer of the outgoing call. The functions instruct the two circuits to withhold initialization and then splice the two circuits.

dktr_call is a support function for **dktr_splice** and **dktr_osplice**. Most applications of **dktr_splice** and **dktr_osplice** do not need to call **dktr_call**; however, it is available as an external function. **dktr_call** makes the outgoing call and allows additional control over the outgoing dialstring. The parameters *baud* will set the baud rate and *rbuf* will set the advertised receive buffer size in the dialstring.

FILES

/usr/lib/libdk.so	host interface subroutine library
/dev/dk/dial	default dialer device
/dev/dk/dial*X*	dialer device for interface *X*
/dev/dk/ctl*X*	*Common Signaling Channel* device for interface *X*

SEE ALSO

dkdial(3X)
dkmgr(3X)
dksplice(3X)
dkurpctl(3X)
dk_uxinfo(3X)

DIAGNOSTICS

dktr_splice and **dktr_osplice** will return zero for a successful request and non-zero upon error. See *dksplice*(3X) for more details about splice errors. **dktr_call** will return a positive number for a successful request and a negative number upon error. Diagnostic messages are written to the file pointed to by *logf*. See *dkmgr*(3X) for more details about *logf*.

WARNINGS

These functions provide access to the CommKit drivers for special applications. They are only supported for data switch end points that have well behaved GOS5 transmitters. These functions should only be invoked while executing under the *root* login name (user ID 0).

NAME

dkurpctl – control the URP initialization of an AT&T data switch connection

SYNOPSIS

dkset_no_ainit(*fd*)
int fd;

dkset_one_ainit(*fd*)
int fd;

extern int dk_verbose;

DESCRIPTION

dkset_no_ainit overides the standard URP initialization processing by blocking all acknowledgements for URP initialization requests. **dkset_one_ainit** overides the standard URP initialization processing by blocking all acknowledgements for URP initialization requests except for the very first request.

FILES

/usr/lib/libdk.so host interface subroutine library

SEE ALSO

dkerr(3X), *dkhs*(7).

DIAGNOSTICS

Both functions will return zero for a successful request and a negative number upon error. Both will print an error message on standard error if the requested action fails, unless the external variable *dk_verbose* has been set to 0 before the call is made.

WARNINGS

These functions provide access to the CommKit drivers for special applications. Its frivolous use may result in unusable connections and other undesirable conditions that will only be cleared by closing the device.

NAME

dkxenviron - transmit variables to a remote host

SYNOPSIS

#include <**dkit/dk.h**>

dkxenviron(*netfd***)**
int netfd;

DESCRIPTION

dkxenviron sends environment variables to a remote host. The variables are sent as a series of null-terminated strings. The environment variables to be sent are listed, separated by commas, in the environment variable DKEXPORT.

The parameter *netfd* is the file descriptor of an open connection to the remote host that is returned by a previous successful *dkdial*(3X)).

DIAGNOSTICS

Upon successful completion a value greater than zero is returned. Otherwise, a value less than zero is returned.

FILES

/usr/lib/libdk.so

/usr/lib/libdk.a

SEE ALSO

dkdial(3X).

NAME

maphost – map destination name to an AT&T data switch dialstring

SYNOPSIS

char *maphost (*host, service, defsufx, defprot, parm***)**
char *host, *defsufx, *defprot, *parm;
char service;

char *miscfield (*service, field***)**
char service;
char field;

DESCRIPTION

maphost maps the destination *host* name into a full data switch address using the file */etc/opt/dk/dkhosts.*

In addition to the *dkhosts* file, the user may create a *.dkhosts* file in his/her home directory (*$HOME/.dkhosts*). **maphost** will map the destination *host* host name using the local *.dkhosts* file first; if the file does not exist or a match is not found, **maphost** will map the host name using the */etc/opt/dk/dkhosts* file. **maphost** references the user's HOME environment variable to build the path to *$HOME/.dkhosts.* The path is built only once during the first call to **maphost**. Therefore, any change to the HOME environment variable on successive **maphost** calls does not change the location of *$HOME/.dkhosts*.

maphost is an auxiliary routine used with *dkdial*(3X). For the **maphost** routine to return a successful match to an entry in *dkhosts*(4), the *host* and *service* class arguments to the **maphost** routine must match the *host* and service *classes* fields, respectively, for some entry in *dkhosts*(4).

If the *host* argument is NULL, **maphost** will return the dialstring for the next host in *dkhosts*(4) that matches the specified *service* class each time the routine is called. Therefore, the first call to **maphost** with a null *host* argument will return the dialstring for the first host in *dkhosts*(4) that matches the *service* class specified. **maphost** does not rewind its file pointer. Therefore, successive calls start searching from the last entry found in either the *$HOME/.dkhosts* file or the */etc/opt/dk/dkhosts* file.

Once a successful call to **maphost** has been made, **miscfield** extracts information from the *Miscellany* column in the *dkhosts*(4) file.

The dialstring is then constructed from the following:

dialstring:	from *dkhosts*(4)
service:	from *defsufx* or from the *Miscellany* field in *dkhosts*(4)
protocol:	from *defprot* or from the *Miscellany* field in *dkhosts*(4)
parameter:	*parm* string to append

FILES

/usr/lib/libdk.so	host interface subroutine library
/etc/opt/dk/dkhosts	host control file for destination mapping
$HOME/.dkhosts	local host control file for destination mapping

SEE ALSO

dkdial(3X), *dkhosts*(4).

DIAGNOSTICS

The **maphost** routine will return a pointer to the dialstring (a static) if there was a match. If there was no match and the *host* argument is not NULL, **maphost** will return a pointer to a character string (a static) containing the value of the *host* argument. If there was no match and the *host* argument is NULL, **maphost** will return a value of NULL.

The **maphost** routine will return a value of NULL if called with null *host* and *service* arguments.

Another item to note is if the host argument contains a full path name (any name with a '/' in it), the **maphost** routine will not search the *dkhosts*(4) file. Instead, it will construct a dialstring using the arguments passed to the **maphost** routine. The **miscfield** routine will return a value of NULL if the *Miscellany* [see *dkhosts*(4)] field is not listed for that particular *host*/*service* combination entry.

NAME

dkacct – host interface accounting file format

SYNOPSIS

#include <**dkit/acct.h**>

DESCRIPTION

Accounting files produced by *dkdaemon*(1M) have records in the form defined by <*dkit/acct.h*> , whose contents are:

```
struct  dkacct
{
        short   dkA_unit;               /* physical interface unit */
        short   dkA_chan;               /* physical channel on unit */
        time_t  dkA_stime;              /* connection start time */
        time_t  dkA_etime;              /* connection end time */
        uid_t   dkA_uid;                /* uid at open */
        dk_chan_stat_t dkA_stats;       /* session statistics */
};
typedef         struct dkacct   dkacct_t;
```

A record is written to the host interface accounting file each time a channel on that interface is closed. Each record describes a complete session including the start and end times in standard *time*(2) format, and the complete transfer statistics. The transfer statistics are defined in <*dkit/rdfp.h*> and contain:

```
struct  dk_chan_stats
{
        u_long xmit_msgs;       /* messages transmitted */
        u_long xmit_blocks;     /* URP blocks transmitted */
        u_long xmit_bytes;      /* bytes transmitted */
        u_long xmit_enq;        /* transmitter ENQuires sent */
        u_long xmit_init;       /* INITn characters transmitted */
        u_long rcv_msgs;        /* messages received */
        u_long rcv_blocks;      /* URP blocks received */
        u_long rcv_badblocks;   /* URP blocks REJected */
        u_long rcv_bytes;       /* bytes received */
        u_long rcv_rej;         /* REJect characters received */
        u_long rcv_enq;         /* ENQuires received */
        u_long rcv_initreq;     /* INITREQ characters received */
        u_long rcv_init;        /* INITn characters received */
};
typedef         struct dk_chan_stats    dk_chan_stat_t;
```

FILES

/var/opt/dk/log/dkacct default accounting file

SEE ALSO

dkdaemon(1M), dkserver(1M), dkstat(1M), dkaudit(4), dkhs(7), dkux(7).
time(2) in the *UNIX System V System Programmer's Reference Manual*.

BUGS

Accounting records for incoming sessions list the user ID of the *dkserver*(1M) that accept the connection rather than the user ID of the spawned process.

NAME

dkaudit – host interface connection auditing record formats

DESCRIPTION

dkaudit logs the activity of the **dkdaemon**(1M) process in terms of server startup attempts (see **SERVER Messages**), outgoing connection requests (see **CONNECTION Messages**), and both inbound and outbound UNIXP messages (see **UNIXP Messages**). These auditing records are generated by the **dkux**(7) when enabled (default) by the **dkux**(7) tunable parameters and the **dkdaemon**(1M) run-time options.

The **dkaudit** records are described below.

UNIXP Messages

These time-stamped messages document `UNIXP` message exchanges between *dkux*(7) and the peer process in the AT&T data switch controller. Both inbound and outbound messages may be recorded:

Month Day HH:MM:SS `UNIXP: (`*Intf*`, `*Chan*`) <=` *12 Hexadecimal Bytes of message*

Month Day HH:MM:SS `UNIXP: (`*Intf*`, `*Chan*`) =>` *12 Hexadecimal Bytes of message*

Intf and *Chan* describe the physical interface and channel targeted by the message; inbound messages are marked with "<=", and outbound with "=>".

Typical messages from a log might look like:

```
Oct  9 16:28:00     UNIXP: (0, 30) <= 03 02 1e 00 00 00 00 00 00 00 00 00

Oct  9 16:47:58     UNIXP: (0, 30) <= 03 01 1e 00 32 00 00 00 00 00 00 00

Oct  9 16:47:58     UNIXP: (0,  1) => 03 01 1e 00 00 00 00 00 00 00 00 00
```

SERVER Messages

dkserver(1M) start-up attempts are recorded as time-stamped SERVER messages. The primary form of the SERVER audit message lists the numerical user ID (*UID*) of the process attempting the DKANNOUNCE server startup request:

Month Day HH:MM:SS `SERVER: (`*Intf*`, `*Chan*`)` *"ServerName"* `Started by UID` *UID*

Month Day HH:MM:SS `SERVER: (`*Intf*`, `*Chan*`)` *"ServerName"* `Failed dk_errno=` *no*`. by UID` *UID*

Successful attempts are marked `Started` and unsuccessful ones `Failed` and include the **dk_errno (no.)**. *ServerName* is the *dkserver*(1M) name provided in the request.

Servers started through *dktli*(7) do not provide a user ID and are recorded without the "`by UID`" information:

Month Day HH:MM:SS `SERVER: (`*Intf*`, `*Chan*`)` *"ServerName"* `Started`

Month Day HH:MM:SS `SERVER: (`*Intf*`, `*Chan*`)` *"ServerName"* `Failed dk_errno =` *no.*

A *dkserver*(1M) started by **root** with the name 'probe' would create the following example audit entry:

```
Oct  9 14:16:54     SERVER: (0,  2) "probe" Started by UID 0
```

CONNECTION Messages

Outgoing connection attempts are recorded as time-stamped CONNECTION messages. The primary form of the CONNECTION audit message lists the numerical user ID (*UID*) of the process attempting the *dkdial*(3X) connection request:

Month Day HH:MM:SS CONNECTION: (*Intf*, *Chan*) *"DialString"* Started by UID *UID*

Month Day HH:MM:SS CONNECTION: (*Intf*, *Chan*) *"DialString"* Failed dk_errno= *no.* by UID *UID*

DialString is the full dialstring sent to the data switch controller, including the generated newline and user ID if used. Successfully established connections are marked Started and failed attempts Failed and include the **dk_errno (no.)**.

Some connections made through *dktli*(7) may not include the "by UID" information:

Month Day HH:MM:SS CONNECTION: (*Intf*, *Chan*) *"DialString"* Started

Month Day HH:MM:SS CONNECTION: (*Intf*, *Chan*) *"DialString"* Failed dk_errno (*no.*)

A failed attempt to print a file on a printer named Bodoni by **root** might appear in the log as:

```
Oct 16 17:15:09 CONNECTION:(0,3) "nj/prt/Bodoni\n0" Failed dk_errno=6 by UID 0
```

A successful connection to the atomic time synchronization service on machine time with parameter sync by user 101 could look like:

```
Oct 10 17:46:45  CONNECTION: (0,4) "time.atomic..sync\n101" Started by UID 101
```

SPLICE Messages

This audit message type is not yet implemented.

Log Level

The table below lists the log levels (controlled by the **-v** option of the **dkdaemon**(1M) command) applicable to the **dkaudit** auditing records along with the information included in the log level.

Log Level	Logged Information
4	Server startups and errors
5	Outgoing connection requests
6	Valid inbound UNIXP messages
7	Outbound UNIXP messages
9	Outbound UNIXP keepalive messages

FILES

/var/opt/dk/log/dkdaemonlog	default *dkdaemon*(1M) auditing log file
/etc/conf/pack.d/dkux/space.c	file containing *dkux*(7) tunable parameters (non-3B2 machines)
/etc/master.d/dkux	file containing *dkux*(7) tunable parameters (3B2 machines)

SEE ALSO

dkdaemon(1M), *dkserver*(1M), *dkerr*(3X), *dkacct*(4), *dksrvlog*(4), *dktli*(7), *dkux*(7).

NAME

dkdotab – transparent remote command control file

DESCRIPTION

This file is used by the *dkdo*(1C) program to obtain information on where and how to process commands. The default file is */etc/opt/dk/dkdotab*.

dkdotab consists of one or more lines, each with exactly five fields. Each time a command emulation is processed by *dkdo*(1C), it scans this file for the first match with the command requested. If a match is found, the remaining fields indicate on which host the command is to be executed, what flags affect the operation of *dkdo*(1C), what option letters are applicable for execution of the desired command, and whether the files associated with this command are input, output, or a combination of both. If a field in the **dkdotab** is not applicable for a given command, a "–" must be specified in place of the field.

The field that participates in the match is the *Command* field. If the call fails to one host, the table is scanned for the next match and the calling process repeated.

The lines consist of the following fields delimited by tabs and / or blanks.

Command: Command to be executed on a remote host. This field length is not to exceed 16 characters.

System: System on which the command is to be executed. This field length is not to exceed 64 characters.

Flags: Zero or more of the flags **s** or **x**. This field length is not to exceed 32 characters.

- **s** If the filenames are preceded by an **s.**, as with SCCS files, the file without the **s.** is sent.
- **x** use **.rx** instead of **.do** as the *service class*. Requesting the **.rx** service on the remote system causes the user's *.profile* on that system to be executed prior to the remote command.

Options: Zero or more options may be specified for a given command. The options correspond to the various command-specific options supported by *dkdo*(1C). This field length is not to exceed 128 characters. The various options are delimited by four possible operators:

- **:** Arguments that follow the option flag are arguments associated with that option. The arguments may or may not follow directly after the option specification.
- **~** Only arguments that directly follow the option flag are associated with that option.
- < Arguments that follow are input files.
- > Arguments that follow are output files.

Files: Description of the type of files that follow the options associated with this command. This field length is not to exceed 16 characters.

- > Output file
- < Input file

***** The preceding file indicator applies to all remaining files associated with this command.

: Preceded by a flag. Insert the indicated flag in the command line before it is invoked.

EXAMPLES

The following table entry exemplifies a valid entry in the **dkdotab.**

lp hosta – **d:n:o:t** – **c:<***

The command **lp** is executed on the system **hosta** [see *dkhosts*(4)] with no flags set. The options indicated are those supported by *dkdo*(1C) for the **lp** command [see *lp*(1)]. The delimiter "**:**" is an indication that the arguments that follow the option flag are arguments associated with that option (the option arguments may directly follow the option letter, or a space may exist between them). The value of the *Files* field – **c:<***, indicates that a flag of – **c** is to be inserted on the *lp*(1) command line on the remote machine before it is invoked and all of the files that follow the options associated with the *lp*(1) command are input files.

FILES

/etc/opt/dk/dkdotab default control table

SEE ALSO

dkdo(1C), *maphost*(3X), *dkhosts*(4).
lp(1) in the *UNIX System V User's Reference Manual.*

NAME

dkgroups – channel group control file

DESCRIPTION

dkgroups is used by the **dkdial**(3X) function to determine the interface and channel numbers to use in making a connection to a remote endpoint. Channel groups are used to restrict users to a specified number of channels and control their access permissions on remote hosts by means of different data switch originating groups. A channel group can be specified by the shell variable DKGROUP or by using the program interface on **dkdial**(3X).

The file consists of one or more lines, each with exactly four tab-separated fields. For each call to **dkdial**(3X) with a channel group, the *dkgroups* file is scanned, stopping at the first match. If a match is found, a connection is attempted using the specified interface(s) and channel range. The call fails if:

1. No match is found
2. Destination rejects the call
3. All channels in the range are in use.

The lines consist of the following tab-separated fields:

```
name      interface      low      high
```

name the name of the channel group; this name does not have to match the originating group name on the data switch.

interface a comma-separated list of interfaces on which to make connection attempts.

low the lowest channel number for the channel group.

high the highest channel number for the channel group.

The channel range consists of the values *low* through *high* inclusive.

EXAMPLES

```
securegrp      1      500      511
```

In this example, a call made by selecting **securegrp** will be attempted on interface **1**, beginning at channel **500** and continuing to channel **511** (for a total of 12 channels). If the connection attempts are unsuccessful in this range, the call will fail.

```
manyintf      1,0      10      20
```

In the above example, calls made by selecting the group **manyintf** will be attempted first on interface **1**, beginning at channel **10** and continuing to **20** (a total of 11 channels), and then on channels **10 - 20** on interface **0**. The data switch originating groups should be the same on both interfaces.

FILES

/etc/opt/dk/dkgroups	channel group control file

SEE ALSO

dkmaint(1M), *dkdial*(3X).

NAME

dkhosts – host control file

DESCRIPTION

dkhosts is used by the *maphost*(3X) function to construct the appropriate AT&T data switch dialstring for the host referenced.

In addition to the **dkhosts** table included with the CommKit Host interface software package (*/etc/opt/dk/dkhosts*), the user may create a **dkhosts** table in his/her home directory (*$HOME/.dkhosts*). If the *$HOME/.dkhosts* file exists, the two *dkhosts* files are treated as one concatenated file when **maphost** is invoked (the local *$HOME/.dkhosts* file followed by */etc/opt/dk/dkhosts*); if the local *$HOME/.dkhosts* file does not exist, the */etc/opt/dk/dkhosts* file is used. The term "*dkhosts* file" refers to both files: *$HOME/.dkhosts* and */etc/opt/dk/dkhosts*. Specific files are designated by the use of the pathnames.

The *dkhosts* file consists of one or more lines, each with exactly four fields. Each time *maphost*(3X) is called, it scans **dkhosts** for the first match with the *Host* and the service *Classes* specified. If a match is found, the remaining fields indicate the appropriate data switch dialstring for that particular host and include miscellaneous information to be parsed. The fields that participate in the match are the *Host* and *Classes* fields.

The lines consist of the following fields delimited by tabs.

Note: Delimit fields by tabs only. Do not use blanks.

Host: Host to which the call is being made.

Classes: Service class to match

- **d** dkdo
- **f** file transfer
- **l** remote login
- **p** printer
- **x** remote execution

Dialstring: Dialstring of host.

Miscellany: Subfields in this field are separated by commas and have two-character names constructed from the service class character and another character appropriate to the service class. The characters representing the field names are:

- **c** command to execute
- **s** service name
- **v** existence of environment variables (y or n)
- **p** protocol string
- **o** old protocol (y or n)

EXAMPLES

An entry of '`ls=rx`' in the *Miscellany* field in the **dkhosts** file, would mean that for *remote* login the *service* invoked is '**rx**'.

An entry of '`fo=y`' in the *Miscellany* field in the **dkhosts** file, would mean that for *file* transfer the *old* file transfer protocol is to be used.

FILES

/etc/opt/dk/dkhosts	host control file for destination mapping

SEE ALSO

authorize(1M), *maphost*(3X).

NAME

dksrvlog – host interface server log file

DESCRIPTION

This file is used by the *dkserver*(1M) program to log information about incoming call requests. The default log file name is */var/opt/dk/log/dksrvlog* and may be changed with the '**-l**' option to *dkserver*(1M). The '**-v**' option to *dkserver*(1M) specifies the amount of information that will be written to the log file. A log level of **1** prints out the least information and a log level of **9** prints out the most information. The default *dkserver*(1M) log level is **6**.

Each entry in the log file (for log levels **6** or lower) begins with a time stamp that includes the date, time, interface, and channel number of the current call. It also contains the process ID of either the *dkserver*(1M) or the current call.

The first message logged after *dkserver*(1M) starts is:

time_stamp **SERVER** *name* **is INITING files=***(srvtab dkuidtab)* **loglvl=***n*

time_stamp **dkmgr: SERVER** *name* **is ACTIVE and SERVING**

where *name* is the name of the host interface server started, *srvtab* is the name of the *dkserver*(1M) control table, *dkuidtab* is the name of the *dkuidtab*(4) file and *n* is the level at which the server activity is logged.

More information is printed depending on the following log levels:

Loglevel	Log Information
1	Each incoming call is logged with the service requested, the user ID, and the dialstring of the remote system the call originated from. The entry will also show whether the call was accepted with the token REQUEST or rejected with the token DENIED.
2	Logs the reasons incoming calls are rejected. The token used is either ERROR or DENIED.
3	Enters information when a call exits, including the device, process ID, and exit code. The exit code is passed from the process which was spawned by the *dkserver*(1M) for the call. Consult the documentation for the exiting process and refer to the *exit*(2), *wait*(2), and *signal*(2) manual pages. The token used is EXIT.
4	Arguments that are passed with the incoming call are logged. The token is ARGS.
5	Same as log level 4.

6 Same as log level 4.

7 Prints information useful for debugging.

8 Logs the number of channels per interface and, for each call, the parsed dialstring. This information is actually logged by the *dkmgr()* function which receives and parses incoming calls. The value of the optional key, DKKEY see *authorize*(1M), is also logged by the *dkmgr()* function.

9 The full dialstring (including DKKEY) as it was received will be logged.

FILES

/var/opt/dk/log/dksrvlog default name of the *dkserver*(1M) log file

SEE ALSO

authorize(1M), *dkserver*(1M), *dkdial*(3X), *dkuidtab*(4), *srvtab*(4).
exit(2), *signal*(2), *wait*(2) in the *AT&T UNIX System V Programmer's Reference Manual*.

NAME

dkuidtab – host server user ID mapping file

DESCRIPTION

This file is used by the *dkserver*(1M) program to obtain information on how to map user ID's from incoming call requests to valid user ID's on the local system. The file consists of one or more lines, each with three fields. Based on control information in *srvtab*(4), *dkserver*(1M) may scan this file for a match of the *Originating Group* name and user ID. If no match is found, *dkserver*(1M) advances to the next control file entry. If a match is found, the remaining fields in **dkuidtab** indicate which local user name and password to use in processing the call request. The default user ID mapping file is */etc/opt/dk/dkuidtab*.

This file is updated by the *authorize*(1M) program. The lines consist of the following fields delimited by blanks:

Field 1: *area/exch/group.user[/key]* –– the *Originating Group* name followed by a period, the remote user ID, and an optional slash followed by a *key*. The *key* contains the value of the DKKEY environment variable at the time of the authorization. The DKKEY variable allows for pre-establishment of more than one login to a remote host from the same login on the local host. See *authorize*(1M).

Field 2: The login name of the user to be used on the local system.

Field 3: The encrypted password of the user on the local system, or a '**:**' if the user does not have a password.

FILES

/etc/opt/dk/dkuidtab	default user ID mapping file
/etc/opt/dk/dkuidtab:o	backup user ID mapping file

SEE ALSO

authorize(1M), *dkserver*(1M), *dkdial*(3X), *dksrvlog*(4), *srvtab*(4).

WARNINGS

The **dkuidtab** file should be checked periodically for corruption. The *authorize*(1M) or *dkserver*(1M) programs only detect one type of corruption in this table and that is whether there are at least three fields on each line. An error is posted in the *dkserver*(1M) log file [see *dksrvlog*(4)] when it detects this error. The *authorize*(1M) program reports the error to the user when it detects this error.

NAME

srvtab – *dkserver*(1M) control file format

DESCRIPTION

This file format is used by the *dkserver*(1M) program to obtain information on how to process incoming call requests. The format consists of one or more lines, each with exactly six fields. The lines are kept in a number of files in the **srvtab** directory. The default directory is */etc/opt/dk/srvtab*. The file to be searched is located by the incoming *Service* name (for example, */etc/opt/dk/srvtab/login* is the name of the file used for the **login** service). If no file exists for a specific service, then the catch-all file "*" is used.

Each time a call is processed by *dkserver*(1M), it scans the appropriate **srvtab** file for the first match with certain fields in the incoming call request information. If no match is found, the call is denied. If a match is found, the remaining fields indicate which program to execute, what arguments to pass to it, etc. The fields which participate in the match are the *System*, *Service*, and *User* fields. The *Flag* field may also be used to limit the scope of the service (see *Flag* below).

Any line that has a leading "#" character is a comment line. All other lines are control lines.

The control lines consist of the following fields delimited by tabs:

Note: Delimit fields by tabs only. Do not use blanks.

System: *Originating Group* name pattern: *area/exchange/group*[!][*.user*]. Originating group patterns may be completely specified, or contain embedded asterisks, question marks, and/or bracketed characters.

Each embedded asterisk matches any set of zero or more characters and each embedded question mark matches any single character. For example, **area6/e*/????** matches only 4-character *groups* in *area* **area6** whose *exchange* begins with the letter **e**; the patterns *****, ***/***, and ***/*/*** are equivalent and match all originating groups.

Brackets are used to match a single character from a list and/or a range of characters. Bracketed characters may be included anywhere in the group pattern, up to but not including the *.user* part of the field. Any ASCII character is valid within brackets. The asterisk (*) and question mark (?) are taken as literal when they appear within brackets.

(**NOTE:** The use of special characters in pattern matching is similar to the use of special characters in **ed**, however it is not exactly the same.)

There are two special symbols which may be used within the brackets to specify a range or the negation of elements: the dash (-) and the caret (^), respectively. The character preceding the –

must be less than or equal to the ASCII value of the character following the –. For example, **area6/exch[1-3]/host0** matches **area6/exch1/host0, area6/exch2/host0**, and **area6/exch3/host0**.

When the ^ symbol is used, it must appear immediately following the left bracket; if it appears in any position *other than* the first position, it is taken as a literal character. The operation of the ^ symbol applies to all characters which follow it. For example, **area6/exch[^1-4]/host0** matches **area6/exch5/host0**. Exchanges **exch1, exch2, exch3,** and **exch4** have been negated by the bracketed expression **[^1-4]**.

If the last character in the pattern is a "**!**", then the pattern does not match the *dkserver*(1M) name. For example, if the local server name is **hostx**, **area6/exch2/*!** matches all groups in *area* **area6** and *exchange* **exch2** except for **area6/exch2/hostx**. This feature can be used to prevent users from calling themselves. **NOTE**: The server name of the host and the *Originating Group* name of its interface on the AT&T data switch must be assigned the same value for the **!** feature to be effective.

Specification of the optional *.user* field allows selection / restriction of services to certain users. The *.user* field is the decimal value of the ID of the user on the originating host. The symbols <, >, and – may be specified in the *.user* suffix. Therfore, the pattern **area1/exchange2/hosta.<uid1** would match any user with an incoming uid *less than* **uid1**. Similarly, using the > would match any user with an incoming uid *greater than* **uid1**.

The hyphen (–) is used to specify a range. For example, the pattern **area1/exchange2/hosta.uid1-uid3** will match any user with an incoming uid between **uid1** and **uid3**, inclusive.

NOTE: When specifying a range, the first *uid* must be less than or equal to the second *uid* in the range.

Service: The specific service requested, (that is, the string which appears after the period in the dialstring). The special entry "–" matches the null service (that is, the case where no specific service is requested). A single asterisk functions as a service wildcard which will match any requested service. The following services exist by convention; others may be added as needed by the application.

-	null service (generally login)
*****	wildcard service (matches any requested service)

login	login
do	invoke commands on remote hosts
rl	remote login
pupu	file transfer (for example, **push/pull**)
rx	remote execution
uucp	uucp handler
authorize	invoke *authorize*(1M) on remote hosts

Flag: One or more of the protocol flags **aehltuvxILPRUTM/|** or **Et** may be specified:

a Additional arguments should be read from the incoming data channel by *dkserver*(1M) before executing the application program.

e Arrange for the exit code of the remotely executed command to be passed back to the originating system.

h Spawn process with SIGHUP ignored.

l Spawn a TLI application from the **srvtab** Service.

t Open the tty mode device driver for the channel, and make it the program's standard input, standard output, and standard error files.

u Use plain URP protocol.

v Environment variables should be read from the incoming data channel by *dkserver*(1M) before executing the proper application program. (Remote execution only.)

x Open the remote execution protocol device driver for the channel.

F Inform *dkserver*(1M) that the invoked program should be a child process of *init*(1M). Use of this flag prohibits the use of any *utmp* related flags: **I, U, L**. If any *utmp* flag is used in conjunction with the **F** flag, the incoming call will be rejected and an error will be written to the *dksrvlog* file.

I Make an INIT_PROCESS type of *utmp*(4) entry. The **I** flag *cannot* be used in conjunction with the **F** flag.

L Make a LOGIN_PROCESS type of *utmp*(4) entry. This flag should be used for all services, like *login*(1) that write the initial entries in the system *wtmp* file. The **L** flag *cannot* be used in conjunction with the **F** flag.

M Use this flag to send mail to the specified user ID when a service is requested illegally.

P Retain the **dkserver**'s *nice*(2) priority when invoking the application. The application is expected to lower the priority after a short interval by means of *nice*(2). If this is not done, incoming call processing can fail because there are too many high priority processes. The **P** flag should *never* be used in a **srvtab** entry that invokes user login processing or any other long running processes.

R Restrict this entry to user IDs having "ordinary" shells (defined as ending in **sh** and not beginning with **r**).

T Trap this call, rejecting it with the error code given in the first entry of the *Parms* field. Also log a message generated by %-substituting the *Parms* field.

U Make a USER_PROCESS type of *utmp*(4) entry. The **U** flag *cannot* be used in conjunction with the **F** flag.

/ If a "/" is present, the flags that precede it are the settings that are used. Any flags that follow the **/** act as the default flag settings but will be replaced by user supplied protocol fields. This flag is invalid if the user supplied protocol field is illegal or if it contains any of the uppercase flags (for example, **I, L, U, R,** or **T**).

| If a "|" is present, the list of STREAMS modules that follows the | will be pushed onto the *stdin* STREAM. The STREAMS module list should be separated by colons (:) if more than one of STREAMS module is specified. The | should be the last flag specified in the flag field.

Et The **E**t flag may appear only in the dialstring as the final two characters of the protocol field. *dkserver*(1M) sets the endpoint type to *t*, overriding the default value provided by the data switch network. The character *t* must be a valid data switch endpoint type code. See *dkepoint*(3X) for additional information.

User: The way in which the calling user ID is to be treated. The user ID may be used unchanged, restricted to a range, mapped using the authorization table, or set to a fixed value.

***n, *o** Use the numeric value of the user ID supplied in the call request information. The ***o** means treat the user ID as octal while the ***n** means treat the user ID as having a self-determining format where an initial **0x** or **0X** indicates hexadecimal, an initial **0** indicates octal, and the presence of neither indicates decimal. If the password for the user ID of the remote user has expired, the entry is considered invalid.

& Translate the supplied origin group name and user ID using the *dkuidtab*(4) file to match users who have established authorization via use of the *authorize*(1M) program. This type of entry matches **only** user IDs which have entries in the *dkuidtab*(4) file. If no entry is found, the *dkserver*(1M) program continues to search the **srvtab** file for a match.

<**uid,** >**uid** The previous two forms can be further restricted to a range of user IDs by appending <*uid* or >*uid* to the field. This restricts the incoming user ID to be less than (greater than) the specified decimal number. For example, ***n**>**0** prevents root (user ID **0**) from using a particular service. Only one modifier may be appended to an entry.

login This type of entry allows a fixed user ID listed in the *passwd*(4) file to be chosen.

= and – The symbols **=** and – are allowed in the User Field to specify a specific *uid* or a range of *uids*, respectively. Examples are shown below:

*n=uid	access granted for a specific uid
*n=uid1-uid2	access granted for a range of uids
*o=uid	access granted for a specific uid
*o=uid1-uid2	access granted for a range of uids
&=uid	access granted for a specific authorized uid
&=uid1-uid2	access granted for a range of authorized uids

Program: The pathname of the program to be executed. This field may contain a **%s** which is substituted for the pathname of the user's shell as obtained from the *passwd*(4) file. Use a "–" to indicate that no program needs to be executed after the incoming call is accepted.

Parms: The initial arguments to the program. If no program is specified in the previous field, enter a – for this field. Arguments separated by colons will be followed by arguments passed on the call, provided that the – **a** flag is present, indicating that additional arguments follow. Arguments may contain tokens which start with a percent sign (**%**) that cause another string to be substituted in their place. Since some older versions of data switches do not support certain features, the substitution string may turn out to be NULL. The following are the possible substitution strings:

%b the baud rate of the calling terminal.

%c Originating channel number (also see **%m** and **%n**).

%d The dialed service name, not including the period or anything after it; (that is, the destination address part of the dialstring).

%e The service field of the dialstring.

%f The *Originating Group* name.

%h The local server name.

%l The originator, as known to local node. **%m** The originating module number (also see **%c** and **%n**).

%n The originating node name (also see **%c** and **%m**).

%o The type of originating device.

%p Parameters from the dialstring, reparsed, so that if they contain colons, separate arguments will be generated.

%r The protocol field of the dialstring, if any.

%s The pathname of the user's shell as obtained from the *passwd*(4) file.

%t The device file name which corresponds to the assigned data switch channel, minus the initial */dev*.

%u The user ID of the user placing the call.

%x The call flag: **F** if this is the first call from the device, **P** if there have been previous calls. Applies only to originating ports which have been assigned a predefined destination.

%z The module type flag will return the module type of the originating device if the data switch includes this information in the dialstring (field 1 of the fifth line of the dialstring).

%C The name of the **srvtab** control directory being used.

%H The originating group name truncated to the length of the host field of an */var/adm/utmp* entry.

%U The name of the user ID mapping file [see *dkuidtab*(4)].

EXAMPLES

The following example illustrates the use of brackets for pattern matching originating group names. The incoming dialstring **arD/exch5/systemb** will match any one of the listed system field entries:

System Field (originating groups):

arD/exch5/system[a-f]
ar[x-zDF]/exch[385]/system[rbv]
ar[^C]/exch5/systemb
arD/exch[^xyz2-4rt6-8]/system[bbbbb]
ar[1D]/ex[xn45]/[s]systemb*
[a-z][r-z]D/exch[861-5]/system[A-b]
?rD/exch?/system[^g-zA-Fa0-7]
arD/[e-f][0-7]/system[?b*]*
arD/exch5/system[xy^b]

The following samples exemplify valid entries in the file */etc/opt/dk/srvtab/rl*, the control file for remote login.

```
# System    Service Flag  User      Program   Initial Parms
#
ar1/exch2/grp?     rl   U/vx  *n          %s         -Dsh
ar1/??ch2/*        rl   U/vx  *n>10       %s         -Dsh
*                  rl   T/vx  *n<10       -          7:Privileged (%u)
ar1/exch6/*.>500   rl   U/vx  &=1000-1700 %s         -Dsh
*                  rl   I/vx  root       /bin/login login
```

The first line matches calls from groups of the form **grp** followed by any single character in exchange **exch2** and area **ar1**. The user making the call must have the same user ID locally and remotely to match the table entry. The second line matches calls from any group whose exchange begins with any 2 characters followed by **ch2** in area **ar1**. In addition, these calls must be originated from "non-administrative" logins (for example, numeric user IDs greater than ten) also using the same user ID locally. The third line traps remote login requests from any users with numerical user IDs less than ten. Their calls are rejected with an ACCESS DENIED code and their user ID is logged in the server log file. The fourth line matches calls with *uid*s greater than 500 from any originating group from area **ar1** in exchange **exch6** which have pre-established authorization to any *uid* in the range 1000-1700. The last entry will invoke the normal *login*(1) program for callers from any area/exchange/group pattern that does not match any of the preceding */etc/opt/dk/srvtab/rl* entries. On UNIX SVR4.2 systems *login*(1) cannot be invoked by *dkserver*(1M) so the last line must be deleted. It could have been replaced with a line using the *ttymon*(1M) process. However, the *ttymon*(1M) process assumes the special file is a STREAMS device and the remote execution driver is not.

This example, from a control file named */etc/opt/dk/srvtab/whoami*, just prints interesting information about the caller:

```
*  whoami /u adm /bin/echo  echo:You are:%u:from:%f:(%n.%m.%c).\n\r
```

The following example, from the control file */etc/opt/dk/srvtab/uucp*, illustrates how to choose different user IDs for UUCP based on the caller:

```
# System     Service  Flag  User   Program   Initial Parms
#
my/mynode/*.5  uucp   u    luucp      %s        uucico
*.?            uucp   u    euucp      %s        uucico
```

```
*.5             uucp    u     ouucp       %s          uucico
```

The first line only accepts calls from user ID five (hopefully a UUCP process) from hosts in the exchange (**my/mynode**) and maps them to the secure login **luucp**. The second line accepts calls from modem-pools (user **?**) and maps them to the "external" login **euucp**. The last line accepts calls from all other multiplexed hosts and maps them to the login **ouucp**.

The following is an example of a trap and mail message to a user "exptools." If the M flag is present, a mail message will be sent to the user "exptools" upon a trap on an illegal service request.

```
# This line is an example of trap & mail message to a user "exptools".
# If the 'M' flag is present a mail message will be sent to the user
#"exptools" upon a trap on an illegal service request.
*   login    M    exptools   /opt/dk/sbin/dksrverr   dksrverr:exptools:\
%e:%f.%u:node=%n, module=%m, channel=%c
```

UNIX SVR4.2 CAVEATS

The terminal login service provides login security on all systems, and can use the **ttymon** port monitor instead of invoking *login*(1) directly. This is required on UNIX SVR4.2 systems because *login*(1) cannot be invoked by the *dkserver*(1M) process. On UNIX SVR4.2 systems, **ttymon** invokes *login*(1) which restricts the minor device number to 0-255. Since the CommKit Host Interface minor number can range from 0-511 for systems with one data switch interface and 0-1023 for systems with two data switch interfaces, the user is restricted to using only minor numbers 0-255 for incoming calls that invoke *ttymon*(1M), i.e., incoming calls that produce a **login:** prompt.

Two options are available to deal with this problem: Limit the total number of CommKit Host Interface channels to a maximum of 256. This is the preferred solution if it is acceptable to have no more than 256 channels available for all CommKit Host Interface applications on the host. If using a system with one CommKit Host Interface, then the interface should be configured to 256 channels or less. If using a system with two CommKit Host Interfaces, then each interface should be configured to 128 channels or less. Select the incoming channel from the originating point for calls that invoke *login*(1). This is the preferred solution if it is necessary to have more than 256 channels available for all CommKit Host Interface applications on the host. The user should select the biggest free channel number less than 256.

If it is assumed that the CPM module to which a CommKit Host Interface 4.0v3 host is connected is in data switch slot 23, then channel 255 can be selected from the originating point as follows:

dkcu area/exchange/host.23.255

The next call attempt on channel 255 will fail and so channel 254 should be selected:

dkcu area/exchange/host.23.254

This type of solution restricts the incoming channel numbers of calls invoking *login*(1) to be less than 256 while allowing other incoming calls, e.g., calls invoking remote execution or TLI, to use largest available channel number, even if that number is greater than 255.

FILES

/etc/opt/dk/srvtab	default **srvtab** directory
/etc/opt/dk/dkuidtab	default user ID mapping file
/var/adm/utmp	Connection accounting

SEE ALSO

authorize(1M), *dkserver*(1M), *dkdial*(3X), *maphost*(3X), *dksrvlog*(4), *dkuidtab*(4).
passwd(4), *utmp*(4) in the *AT&T UNIX System V System Administrator's Reference Manual*.
login(1) in the *AT&T UNIX System V User's Reference Manual*.

WARNINGS

A single flat file format is still supported but is discouraged because of its impact on performance. In the flat file format the lines of the file are identical to the ones in the files named after the services. The default flat file is */etc/opt/dk/srvtab*.

NAME

dkhs – High-Speed *CommKit* Host Interface STREAMS driver

DESCRIPTION

dkhs is a STREAMS device driver that provides a high-performance *CommKit* Host Interface for the host system. This driver handles all URP protocol encoding and packet assembly, leaving all call establishment functions to be performed by the optionally pushed *dkux*(7) module.

The *dkdaemon*(1M) process started when entering *init* state 2 by the *dkitrc*(1M) script opens the Stream associated with each *Common Signaling Channel* device and pushes and holds the *dkux*(7) module on the Stream for support of the interfaces. Each physical interface remains active as long as *dkdaemon*(1M) keeps the *dkux*(7) module pushed on the associated *Common Signaling Channel*.

Read-side Behavior

Several different STREAMS messages are generated by the **dkhs** driver:

`M_CTL` Supervisory interactions between this driver and the *dkux*(7) call processing module take place via commands contained in `M_CTL` messages. Supervisory messages are sent upstream only on Streams associated with the *Common Signaling Channel* devices where it is expected that they will be processed by the *dkux*(7) module.

`M_DATA` Inbound received data is sent upstream by this driver encapsulated in `M_DATA` messages. Each `M_DATA` message chain consists of a single user-level message of normal data and should not be interpreted as URP control codes or commands.

Received messages are split at intermediate (that is, `BOTM`) URP data block boundaries whenever necessary to prevent an accumulated message from exceeding the `dkhs_rcv_max_msg_sz` tunable parameter. Messages are also split at every point an URP Level–D control code is encountered in the data stream.

`M_FLUSH` Standard STREAMS support for `M_FLUSH` messages is provided.

`M_HANGUP` **dkhs** sends an `M_HANGUP` message upstream on the Stream associated with a circuit when the *dkux*(7) module indicates via a supervisory message that the circuit has been disconnected at the remote end or by the network.

`M_IOCACK` All successfully processed `M_IOCTL` messages are positively acknowledged with an upstream `M_IOCACK` message.

`M_IOCNAK` All unsuccessfully completed or unrecognized `M_IOCTL` requests are negatively acknowledged with upstream `M_IOCNAK` messages.

`M_PCSIG` Interrupts from the *CommKit* Interface hardware are converted into `M_PCSIG` messages when the physical interface is in diagnostic mode. These messages are sent upstream on the Stream associated with the diagnostic device for the interrupting interface.

`M_PROTO` URP Level–D control codes received for a circuit are sent upstream in `M_PROTO` messages.

`M_SETOPTS` The Stream-head for each **dkhs** device is configured on initial open for message no-discard mode to preserve read message boundaries.

Write-side Behavior

Various STREAMS messages are recognized and processed by the **dkhs** driver:

`M_BREAK` The URP Level–D control code for `BREAK` is transmitted on the circuit associated with the Stream when this message is received.

`M_CTL` Supervisory interactions between this driver and cooperating upstream modules and drivers (that is, *dkux*(7), *dktli*(7), etc.) take place via commands contained in `M_CTL` messages.

`M_DATA` Outbound data for transmission should be sent to the driver encapsulated in `M_DATA` messages.

The message chain is transmitted on the circuit using URP GOS5 as zero or more intermediate `BOTM` blocks followed by a single `BOT` block.

`M_DELAY` The appropriate combination of URP Level–D `DELAY` control codes is transmitted on the circuit associated with the Stream when this message is received. The one or more control codes are transmitted within a single URP block.

`M_FLUSH` Standard STREAMS support for `M_FLUSH` messages is provided.

`M_IOCTL` A number of commands may be sent from user-level to the driver in the form of `M_IOCTL` messages.

`M_PROTO` The transmission of URP Level–D control codes may be requested via appropriately formatted `M_PROTO` messages.

`M_START` Starts the previously halted transmitter associated with the Stream so user data transmission on the circuit may resume.

`M_STARTI` Releases the previously flow-controlled receiver associated with the Stream so incoming data may be sent upstream.

`M_STOP` Halts the transmitter associated with the Stream so no new user data blocks are transmitted on the circuit until a subsequent `M_START` is received.

`M_STOPI` Flow controls the receiver associated with the Stream so no `M_DATA` messages are sent upstream until a subsequent `M_STARTI` is received.

FILES

/dev/dk/ctl*X*	*Common Signaling Channel* device for interface *X*
/dev/dk/diag*X*	diagnostic device for interface *X*
/dev/dk/dial	default dialing device
/dev/dk/dial*X*	clone device for interface *X*
/dev/dk/*intf.chan*	individual raw channel devices

SEE ALSO

dkdaemon(1M), *dkdiag*(1M), *dkitrc*(1M), *dkmaint*(1M), *dkserver*(1M), *dkleveld*(3X), *dktli*(7), *dkty*(7), *dkux*(7).
getmsg(2), *putmsg*(2), *read*(2) in the *UNIX System V Programmer's Reference Manual.*

DIAGNOSTICS

Several error values have different or uncommon meanings when received from the **dkhs** driver. The errors returned by **dkhs** are:

`EBADMSG` A *read*(2) system call will return this error when a `M_PROTO` message containing an URP Level–D control code reaches the head of the Stream. Use *isdkleveld()* (see *dkleveld*(3X)) or *getmsg*(2) to retrieve the message containing the Level–D code.

`EBUSY` All attempts to open a device that is marked EXCLUSIVE or attempts to open the diagnostic device while the interface is active will be failed with an `EBUSY` error return code.

`EHOSTDOWN` Opens of the clone device or the individual channel devices will be rejected when *dkdaemon*(1M) is not active or when the daemon is in the process of reinitializing the interface. Opens of the *Common Signaling Channel* device will be rejected with this error when the data link between the *CommKit* Interface hardware and the CPM-HS cannot be activated.

`EIO` Attempts to open the diagnostic device will be rejected with the `EIO` error when a fault occurs entering diagnostic mode. Consult the system console or console logs for additional diagnostic messages.

`ENODEV` Opens of non-existent devices and interfaces will be rejected with this error code. Attempts to open interface devices after a configuration failure will also result in `ENODEV` error return codes.

NAME

dkmx – remote execution multiplexer

DESCRIPTION

dkmx is a STREAMS multiplexing driver that links the remote execution user interface [see *dkxqt*(7)] with the host interface raw driver [see *dkhs*(7)]. The driver has a single device */dev/dk/dkxmx0* that is opened by the *dkdaemon*(1M) process. *dkdaemon*(1M) processes the driver's requests for links to the *dkhs*(7) driver.

This interface is used internally to support the remote execution protocol [see *dk*(1C)] and does not support any user-level application. An open by any process that is not a *dkdaemon*(1M) process will have insipid but benign results.

FILES

/dev/dk/dkxmx0 *Common Signaling Channel* device for the interface

SEE ALSO

dk(1C), *dkdaemon*(1M), *dkhs*(7), *dkxqt*(7).

DIAGNOSTICS

The driver sends a single log message to *dkdaemon*(1M) at startup time. Messages like the following will be entered into the daemon log file:

```
Feb  4 17:46:28     LOG: (0,  0) dkxqt mux driver is active
```

Error values have the following meanings when received from the **dkmx** driver:

`EBUSY` Attempts to open the driver a second time will fail with this error.

`EPERM` Attempts to open the driver by a process that does not have superuser permissions will fail with this error.

`ENXIO` Attempts to open a device other than */dev/dk/dkxmx0* will fail with this error.

`EAGAIN` If there is insufficient memory, a open will fail with this error.

NAME

dkpe – *CommKit* Host Interface Board driver

DESCRIPTION

dkpe is a character special device driver that provides a high-performance interface between the *CommKit* interface hardware and the *dkhs*(7) driver. This driver supports downloading, queue management and interrupt vectoring for all *CommKit* Host Interfaces resident in the system.

Access to the **dkpe** driver is through the *dkipump*(1M) command and the *dkhs*(7) driver.

FILES

/dev/dkpe*X*	download device for interface *X*

SEE ALSO

dkipump(1M), *dkhs*(7).

NAME

dktli – *CommKit* Interface Connection Oriented Transport Provider

SYNOPSIS

t = t_open(/dev/dktp0","O_RDWR, NULL);

DESCRIPTION

dktli is the *CommKit* Host Transport Provider that supports a Transport Provider Interface (TPI). Programs can access **dktli** using the Transport Level Interface (TLI) [see *UNIX System V Release 4.0 Programmer's Guide: Networking Interfaces* for more information about TLI] where it supports the connection-oriented with orderly release (T_COTS_ORD) service type. **dktli** supports all TLI connection-mode primitives and all TLI common management primitives except *t_optmgmt*. **dktli** does not support any connectionless mode primitives and does not support transmission of user data during any state other than data transfer.

dktli consists of the **dktm** multiplexing driver. **dktli** interfaces with the *dkux*(7) module (which provides call management functions) and *dkhs*(7) driver at the bottom for call management and data transfer. **dktli** uses a user-level daemon process *dkdaemon*(1M) to link and unlink *dkhs*(7) Streams below the multiplexing driver **dktm .** The TPI becomes available when the *dkdaemon*(1M) process starts and the TPI becomes inaccessible when the *dkdaemon*(1M) process stops.

The *net_spec* associated with **dktli** is *dktpX*, where *X* is the host hardware interface number. On systems installed with more than one host hardware interface, application programs may access a particular hardware interface through invoking *t_open*(3N) with the desired *net_spec* (for example, **/dev/dktp0**) as the argument. A network server may offer a service with the name, LISTENAME, known to its clients by invoking *t_bind*(3N) to bind LISTENAME to the transport endpoint.

EXAMPLE

The code fragments below illustrate how a client can establish an AT&T data switch circuit through **dktli** to a server listening on address LISTENAME.

```
#include  <tiuser.h>

     struct  t_call  *sndcall;

     fd = t_open("/dev/dktp0", O_RDWR, NULL);
     t_bind(fd, NULL, NULL);
          .
          .
          .
     sndcall->addr.buf = "area/exchange/LISTENAME";
     sndcall->addr.len = sizeof("area/exchange/LISTENAME");
     t_connect(fd, sndcall, NULL);
```

FILES

/dev/dktp*X*	control device for interface *X*
/usr/lib/libnsl.so	TLI subroutine library

SEE ALSO

dkdaemon(1M), *dkhs*(7), *dkux*(7).
listen(1M), *nlsadmin*(1M), *rfadmin*(1M), *sacadm*(1M), *strace*(1M), *sterr*(1M), *t_bind*(3N), *t_open*(3N), *netconfig*(4), *timod*(7), *tirdwr*(7), in the *UNIX System V System Administrator's Reference Manual* and *UNIX System V Release 4.0 Programmer's Guide: Network Interfaces*.
log(7) in the *UNIX System V Programmer's Guide: STREAMS*.

DIAGNOSTICS

The STREAMS log routine *strlog()* is used to log trace and error messages.

NAME

dkty – host tty interface module

DESCRIPTION

dkty is a STREAMS module that provides an interface between the *ldterm*(7) standard terminal STREAMS module and the *CommKit* Host Interface STREAMS driver *dkhs*(7). Its primary function is to intercept and translate write-side terminal *ioctl*s(2) and read-side URP level-D code messages.

The functions in the *c_cflag* field of the *termio*(7) / *termios*(2) structure that are performed by the **dkty** module are PARENB, PARODD and B0. The other functions are ignored by the **dkty** module. (Note that the default flags set by the *dkserver(1M)* are B9600, CS7, CREAD and HUPCL. Therefore, the default is that the parity generation is disabled.) If parity generation is enabled and CS8 is not set, the PARODD flag specifies odd parity if set; otherwise, even parity is used. If the baud rate field CBAUD is set to zero (B0), the **dkty** module will disconnect the line by sending an M_HANGUP message to the Stream-head.

The **dkty** module does not perform the functions specified by the INGPAR, PARMRK, and INPCK flags in the *c_iflag* word of the *termio*(7) / *termios*(2) structure. In general, input parity is ignored and all eight bits are passed up to the STREAM module above (such as, **ldterm**).

Read-side Behavior

The **dkty** module scans all upstream `M_PROTO` messages for URP Level–D codes and translates them appropriately. All other messages and `M_PROTO` messages that do not contain Level–D codes are passed upstream unchanged.

`M_PROTO` Messages containing an URP Level–D `BREAK` code are converted into an upstream `M_BREAK` message. Messages containing a URP Level–D `EOF` code are converted into a zero length upstream `M_DATA` message. `M_PROTO` messages containing any other URP Level–D codes are squelched.

Write-side Behavior

Two downstream STREAMS messages are recognized and processed by the **dkty** module:

`M_CTL` All downstream `M_CTL` messages are silently freed.

`M_IOCTL` A number of commands may be sent from user-level processes to the module in the form of `M_IOCTL` messages. These commands and requests are described below.

IOCTLS

The following *ioctl*s(2) are processed by the **dkty** module. All other *ioctl*s(2) are forwarded downstream. These *ioctl*s(2) are provided for compatibility with the standard terminal interface.

`TCGETS` The argument is a pointer to a *termios*(2) structure. The current value of the *c_cflag* is set and returned.

`TCSETS` The argument is a pointer to a *termios*(2) structure. The value of the *c_cflag* is used to set the terminal parameters. If the value of CBAUD is zero, an M_HANGUP message is sent upstream.

`TCSETSW` The argument is a pointer to a *termios*(2) structure. The value of the *c_cflag* is used to set the terminal parameters. If the value of CBAUD is zero, an M_HANGUP message is sent upstream.

`TCSETSF` The argument is a pointer to a *termios*(2) structure. An M_FLUSH message is sent downstream and the value of the *c_cflag* is used to set the terminal parameters. If the value of CBAUD is zero, an M_HANGUP message is sent upstream.

`TCGETA` The argument is a pointer to a *termio*(7) structure. The current value of the *c_cflag* is set and returned.

`TCSETA` The argument is a pointer to a *termio*(7) structure. The value of the *c_cflag* is used to set the terminal parameters. If the value of CBAUD is zero, an M_HANGUP message is sent upstream.

`TCSETAW` The argument is a pointer to a *termio*(7) structure. The value of the *c_cflag* is used to set the terminal parameters. If the value of CBAUD is zero, an M_HANGUP message is sent upstream.

`TCSETAF` The argument is a pointer to a *termio*(7) structure. An M_FLUSH message is sent downstream and the value of the *c_cflag* is used to set the terminal parameters. If the value of CBAUD is zero an M_HANGUP message is sent upstream.

`JWINSIZE` The argument is a pointer to a **jwinsize** structure. If each element of the current window size structure is zero, the error EINVAL is returned. Otherwise, the current window size structure is copied to the data area of the message block and returned.

`TIOCGWINSZ` The argument is a pointer to a **winsize** structure. If each element of the current window size structure is zero, the error EINVAL is returned. Otherwise, the current window size structure is copied to the data area of the message block and returned.

`TIOCSWINSZ` The argument is a pointer to a **winsize** structure. If the window size that is being set is different from the current window size, a SIGWINCH is sent upstream. The current window size structure is then set from the data area of the message block.

`TIOCSTI` The argument is a pointer to a **char**. The one byte data is copied to a one byte message block, and the one byte message block is sent upstream.

SEE ALSO

dkhs(7).
ioctl(2), *termios*(2) in the *UNIX System V Programmer's Reference Manual.*
ldterm(7), *termio*(7), *ttcompat*(7) in the *UNIX System V System Administrator's Reference Manual.*

DIAGNOSTICS

The **dkty** module returns errors on an open or an *ioctl*(2) failure. These errors are:

`ENXIO` Returned on all failed open attempts. Can occur if the **dkty** open routine is called without the MODOPEN flag set or if one or more STREAMS resources is exhausted.

`EINVAL` This error is returned when all elements of the window size structure are zero for a `JWINSIZE` or a `TIOCGWINSZ` *ioctl*(2) request.

`EAGAIN` This error is returned when a message block allocation has failed.

`EPROTO` Improper *ioctl*(2) requests are returned with this error.

NAME

dkux – *CommKit* Host Interface call processing module

DESCRIPTION

dkux is a STREAMS module that provides call processing support for the host interfaces by communicating with peer processes in the AT&T data switch controller via the *dkhs*(7) driver. Call processing and initialization are accomplished by means of messages passed between the data switch controller and the host, the format and content of which are defined by the UNIXP protocol. **dkux** contains an implementation of the UNIXP protocol.

dkux performs bookkeeping functions relating to the status of data switch circuits on each interface. It must also continually exchange special messages with the controller to inform the controller that the host is active. To ensure that **dkux** is always present, the *dkdaemon*(1M) process started when entering *init* state 2 by the *dkitrc*(1M) script opens the Stream associated with the *Common Signaling Channel* device and pushes and holds the **dkux** module on the Stream. The interface remains active as long as the *dkdaemon*(1M) keeps **dkux** pushed on the *Common Signaling Channel*.

There are also functions performed by **dkux ,** such as call setup, which are required on an as-needed basis. To this end, the **dkux** module may be pushed on any Stream associated with a data switch circuit.

Read-side Behavior

The following STREAMS messages are acted on by the **dkux** module when received from downstream:

`M_CTL` Supervisory interactions between this module and the *dkhs*(7) driver take place via commands contained in `M_CTL` messages. These messages are processed when received from downstream on the Stream associated with the *Common Signaling Channel*. On other channels, they are passed upstream uninterpreted.

`M_DATA` Messages received from the data switch controller arrive on the *Common Signaling Channel* packaged in `M_DATA` messages. **dkux** processes these messages according to the UNIXP protocol. `M_DATA` messages received on other channels are passed upstream uninterpreted.

`M_FLUSH` Standard STREAMS processing for `M_FLUSH` messages is supported in the **dkux** module.

Messages of all other types are passed upstream uninterpreted.

The following STREAMS messages are generated by the **dkux** module and sent upstream:

`M_IOCACK` All recognized and successfully processed `M_IOCTL` messages are positively acknowledged with an upstream `M_IOCACK` message.

M_IOCNAK All recognized but rejected M_IOCTL requests are negatively acknowledged with upstream M_IOCNAK messages.

Write-side Behavior

The following STREAMS messages are recognized and processed by the **dkux** module when received from upstream:

M_IOCTL A number of commands may be sent from user-level to the driver in the form of M_IOCTL messages.

M_FLUSH Standard STREAMS processing for M_FLUSH messages is supported in the **dkux** module.

The following message types are generated by **dkux** and are sent downstream.

M_CTL Supervisory circuit control information is passed down to the *dkhs*(7) driver in M_CTL messages. These messages are sent downstream on the Stream associated with the *Common Signaling Channel*.

M_STOPI To minimize data loss during a splice attempt, the **dkux** module suspends the **URP** receivers by sending M_STOPI messages downstream on the Streams associated with the circuits to be spliced.

M_STARTI If a splice attempt cannot be completed, **dkux** resumes the **URP** receivers by sending M_STARTI messages downstream on the Streams associated with the circuits that were to be spliced.

M_DATA Messages destined for the data switch controller are sent downstream in M_DATA messages.

M_FLUSH When necessary, **dkux** sends M_FLUSH messages downstream on the *Common Signaling Channel* to cause the *dkhs*(7) driver to flush any queued UNIXP messages.

SEE ALSO

dkdaemon(1M), *dkitrc*(1M), *dkhs*(7).
ioctl(2) in the *UNIX system V Programmer's Reference Manual.*

DIAGNOSTICS

The **dkux** module uses the following error values when rejecting *ioctl*(2) attempts. General interpretations for these error values are presented below:

EINVAL The format or content of the *ioctl*(2) request is improper.

EBADF When processing a DKSPLICE *ioctl*(2) it is found that the previous DKSPLICEPREP *ioctl*(2) did not reference the same circuits indicated by the DKSPLICE *ioctl*(2)

EBUSY	A resource required for completion of the *ioctl*(2) call is not available. This does not apply to an inability to allocate buffers.
ENOSPC	**dkux** is unable to allocate a buffer which is needed in the course of processing the *ioctl*(2) request.
EIO	A software error of unknown cause occurred while processing the *ioctl*(2) request.
ENODEV	Circuits involved in the *ioctl*(2) call are not in the proper state.
ENOENT	The *Common Signaling Channel* has not been configured and, thus, the interface is unavailable.
ENOMEM	Circuits involved in the *ioctl*(2) call are not in the proper state.
EPERM	The user who issued the *ioctl*(2) call does not have super-user privileges.

NAME

dkxqt – remote execution character special device

DESCRIPTION

dkxqt is a character special device that provides processes with an interface that implements the remote execution protocol of the *CommKit* Host Interface. This interface is used on the remote end of a remote login or remote execution circuit. It cannot be used to originate calls.

The routines within the **dkxqt** driver are the following:

dkxopen() Opens a channel of an incoming call for remote execution.

dkxclose() Closes and takes down the channel for remote execution.

dkxread() Reads from the calling process. This routine puts together a read request, sends it to the calling host and waits for a reply indicating the amount of real data to be received over the channel. It then reads in the data and returns to the user process.

dkxwrite() Writes to the calling process. This routine first sends a write request, then sends the data to the calling host. It then waits for a reply message indicating the amount of data written. It returns this amount to the user process.

dkxioctl() Send ioctl commands to the calling host. This routine sends an ioctl request followed by the data to the calling host and waits for a response. It also reads any data transmitted from the calling host and returns this to the user process.

FILES

/dev/dkx/xqt	*Common Signaling Channel* device for the interface
/dev/dkx/*intf***.***chan*	individual remote execution devices

SEE ALSO

dk(1C), *dkdaemon*(1M), *dkhs*(7), *dkxmx*(7).

DIAGNOSTICS

Error values have the following meanings when received from the **dkxqt** driver:

`EIO` Attempts to open a device when the interface is not active or when the device is experiencing problems will fail with this error. Reads, writes, and ioctls will also return this error for device problems.

`ENXIO` Attempts to open minor device 0 or 1 will fail with this error.

`EAGAIN` Attempts to reopen a device that is still cleaning up after the last close will fail with this error.

WARNINGS

The calling *CommKit* host must use the *dk*(1C) command for writing a block of data over a channel using the remote execution protocol or the results are unpredictable. The protocol is overhead-intensive and can be extremely slow in a WAN.

Index

D

E

F

G

H

I

J

L

M

N

O

P

R

S

T

U

V

W