TransSys™ DialUp-IP
for the NeXT platform

info@TransSys.COM

Some
Release Notes and Documentation

January 1993

Acknowledgements

This software package contains code from two sources. The first is from the CSNET/CREN DialUpIP software package which states:

Other software in the package is derived from code which was written by the University of California, Berkeley, and carries this notice:

What this software package does.

The TransSys™[1] Dial-Up IP software allows your NeXT computer to be attached to a TCP/IP based network using the serial ports as the network interface, rather than the Ethernet interface. Once a network connection is established between your NeXT computer and the remote network server or router, it is possible to use the connection to support multiple simultaneous connections for remote login, file transfer, mail, or and other TCP/IP based network protocol.

Of course, a 9.6 kb/s serial link isn't nearly as fast as a 10Mb/s Ethernet port. You'll probably not want to try to use bandwidth intensive applications, like NFS (the Network File System).

This software is usually used by a end-system which dials into some network server which also expects to provide network connections over serial links. This software contains a kernel or operating system level driver that transports IP datagrams across the serial link using a trivial encapsulation called **SLIP**[2] (Serial Line IP). You have to be talking to another device which also knows how to do **SLIP**, and not just a terminal server.

This software is also capable of implementing Van Jacobson-style TCP Header compression, which is a technique to vastly improve interactive response time for TCP connection across the SLIP link by *compressing* or eliminating redundant header information. This is commonly referred to as **CSLIP**[3]

One of the unique capabilities of the CREN/CSNET implementation of SLIP called DialUpIP, which this NeXT version is based on, is that a companion user-level daemon is present to automatically dial-up and take down the SLIP link on demand.

---

[1]**TransSys** is a trandemark of TransSys, Inc.
[2] See RFC-1055, "*A NONSTANDARD FOR TRANSMISSION OF IP DATAGRAMS OVER SERIAL LINES: SLIP*", J. Romkey, June 1988
[3] See RFC-1144, "*Compressing TCP/IP Headers for Low-Speed Serial Links"*, V. Jacobson, LBL, February 1990.

<u>What is provided as part of this package</u>

There are two versions of this software; one is available free of charge from various NeXT anonymous FTP software distribution sites as well as via other channels. This version implements the basic SLIP capability, along with the dial-on-demand feature.

Another version also implements CSLIP. The "free" version also includes a demonstration version of the CSLIP capability which can be "test driven." This version is will available for a modest fee. In the mean time, the demo version can be used for a limited amount of time after your NeXT is booted; if you can make it work to your satisfaction, and be convinced that its worth some money to have the capability, then you can buy a full-time license for it. There will be no suprises or disappointments *after* you've spent your money.

This software package is mainly aimed at people which have some experience configuring and installing IP networks and hosts. It is also possible to buy supported versions of this software from commercial vendors which have sub-licensed the package and turned it into a real product.. They can offer a more traditional "product" with telephone support, improved manuals, point-and-shoot installation tools, etc. If you require more detailed assistance or are unfamiliar with SLIP and networking, this is a distribution channel that you should consider.

This document is finally getting too large and unwieldy for WriteNow. It suffers from lack of a table of contents or index, neither of which can be automatically maintained by WriteNow. Perhaps a future version will be in WordPerfect, except that the only portable form of the document would be PostScript since not everyone has WordPerfect, and WordPerfect won't make RTF. Sigh...

How to use this software

This software is most likely used in the situation where the NeXT system is at some remote site, and dials into a "SLIP server" host, which might serve multiple remote sites, in order to obtain a network connection. This common configuration can be illustrated as:

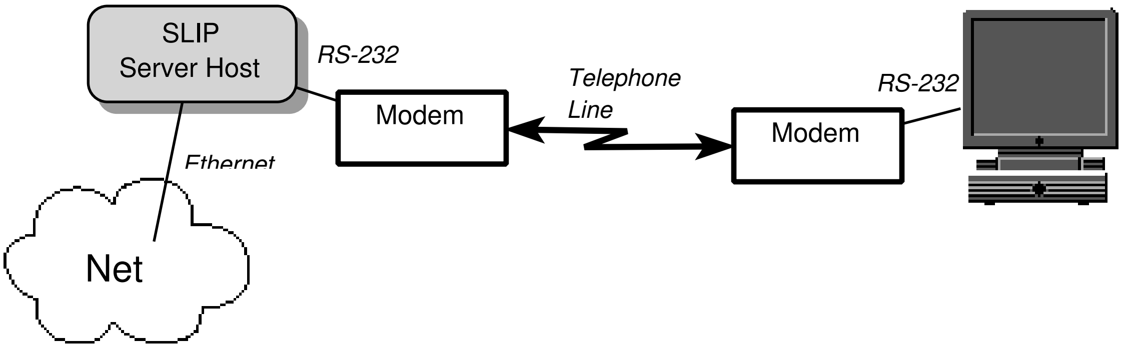

Figure 1

The remote NeXT system (depicted on the right) makes a connection as required using a modem to a SLIP server host which terminates the other end of the link. The SLIP server host presumably also has a connection to a local network (Ethernet, etc) and routes the traffic from the remote NeXT system to that attached network, and beyond.

Of course, many other configurations are possible. In fact, the NeXT can act as the "SLIP Server Host" with this software as well.

How to get started

There are a number of activities which need to be correctly performed to make use of this software on your NeXT computer system. They fall into a few major areas:

• Hardware configuration: connecting your modem to your NeXT computer system correctly.

• Network configuration: collecting the proper network configuration and routing information to enable your NeXT computer to operate correctly on the SLIP link.

• Modem specific configuration: dialing scripts to cause your modem to correctly configure itself and dial the remote end of the SLIP connection.

Warning message from kernel driver

When the kernel driver is loaded, you may see a message logged of the form:

```
dialupip: Can't patch tcp_output() MSS option bug: unknown kernel version
dialupip: Note that this is NOT a fatal error.  Don't panic.  See manual
dialupip: for further details about this issue.
```

It is important that you understand the meaning of this message so that you don't needlessly worry about problem that you may or may not have. The kernel driver is attempting to patch around a problem reported to NeXT in the tcp_output() kernel function.

It relates to an error where a TCP maximum segment size (MSS) option is not sent if the MTU is smaller than 576 bytes. If you are using a SLIP connection with both ends agreeing to use a smaller value, then needless IP fragmentation will take place. This introduces considerable extra space overhead. This situation usually occurs only if you are using TCP header compression with smaller MTUs on on order of 256 bytes.

The kernel driver has a table of patch locations that corresponds to various versions of the NeXT kernel. If the message is logged, then it means that the driver does not have patch locations for your version of the NeXT kernel.

It is unfortunate that this even needs to be done, as this bug has been reported to NeXT in the 2.2 Release and the 3.0 PR1 and 3.0 PR2 releases, along with source code fixes. Perhaps when NeXT gets around to changing the 3 lines of code necessary to repair this problem, all this will be unnecessary. It is not clear to me why after 3 reports of this bug, it has gone unfixed for so long. If you find out what the problem is, be sure to let me know.

Hardware Configuration

It is very important that you correctly connect the modem (or modems, if you want to use more than one at a time) to your NeXT system. The software in the DialUp-IP system requires that all of the modem-control signals at the modem are properly connected to the correct pins on the serial port.

Minimally, you should ensure that the Ground, Transmit Data (TXD), Receive Data (RXD), Data Terminal Ready (DTR) and Carrier Detect (DCD) are properly wired. If you want to use hardware flow control between your modem and NeXT system (and this ***is*** recommended if your modem supports it), then you also need to make sure that Request To Send (RTS) and Clear To Send (CTS) are also correctly wired.

For details of how such a cable should be wired, you should consult the **zs(4)** UNIX manual page, or refer to the printed copy of that manual page reproduced in an appendix in the *Network and System Administration* manual provided with your NeXT computer system.

Please note that Macintosh™ serial cables **will not** work correctly for this application. Also, you should be running Release 2.1 of the NeXT system software on your NeXT as this version contains bug fixes in the serial port kernel drivers which fix some problems in Release 2.0.

For dial-on-demand use, you should make sure that your modem correctly asserts DCD (Carrier Detect) only when a connection is actually made, rather than having it asserted at all time. In a similar fashion, ensure that DTR (Data Terminal Ready) is **not** ignored by the modem; it is important that the modem hang up the phone and terminate an existing call if DTR is dropped by the NeXT.

The DialUp-IP software will work on both Motorola 68030 and Motorola 68040 based NeXT computer systems. The only concern for the 68030 based systems is a consideration of the maximum reliable speed that the serial ports can be used at. On 68030 system, reliable operation is likely not possible at speeds greater than 9600 bps. On 68040 systems, using hardware flow control, operation at speeds as high as 38400 bps is possible.

Network Configuration

Probably the most difficult part of installing and operating this (or any other networking software) on your NeXT is getting the network configuration correct. Most of the effort needs to be expended to make sure that the network *routing* is correct. Before we get into that too deeply, let's take a step back and briefly review how IP (Internet Protocol) networking works.

Hosts on an IP based network communicate with each other by sending small units of data, called ***packets*** to each other. Each packet (also called a *datagram* in the IP world) is a self-contained unit as it passes across a network between computer systems. It is labeled with the *destination* IP address of where it's going, a *source* IP address that describes which host it originated from and other information not within the scope of this discussion. It also, of course, carries the actual data that you are interested in moving around across the network.

The source and destination IP addresses are of particular concern to us when configuring any host on an IP network. As the packet traverses the network, routers along the way examine the destination address and then perform a *routing* function to determine which way to send the packet toward its destination. End system hosts, such as your NeXT also contain a *routing table* which describes the possible alternative routes that a packet might be transmitted to get to its destination.

Why does a host need a routing table? Two major reasons:

• If your host is on an Ethernet network, it might have more than one *router* on the network, each of which is the "next hop" for a certain group of destination networks.

• If your host has more than one network interface (such as an Ethernet interface, and one or more SLIP interfaces), the route selects which interface is to be used for a particular destination.

When configuring the routing for your SLIP connection, you must ensure that routes on your host are installed to correctly route traffic over the SLIP connection when that's appropriate. In the "normal" case, where the NeXT workstation is stand-alone at the remote site, it is assumed that the **default** route for all non-local (*e.g.,* on the same host) traffic is over the SLIP connection. The sample `/usr/dialupip/config/rc.slip` and `/usr/dialupip/config/config.slip` files reflects this, as it installs a default route which points out over the `slip0` SLIP interface.

The ***NetManager*** *application: or the road to Hell is paved with good intentions.* (Of course, this is just *my* opinion...)

Each operating network interface on your host needs to be configured; that is, you set certain values such as the IP address, subnet mask, broadcast address (for Ethernet interfaces) or remote address (for point-to-point network interfaces like SLIP). When an address is configured, the UNIX kernel normally inserts an entry in the routing table which corresponds to the addresses that you configured the network interface with. **This is why it is a very bad idea to configure the Ethernet interface with the same address as one or more SLIP interfaces.** While it is possible to do so, you must "know" what the UNIX kernel does and work around it to make it work successfully.

Now, the **NetManager** application provided with your NeXT is used to configure the address and other characteristics of the **Ethernet interface only**, and should not be used to set any addresses or other configuration information for the SLIP interface(s) on your system. If you wish, you can use **NetManager** to configure the Hostname of your system. If you are using your NeXT stand-alone at a remote site and you do not have a local ethernet network, you should make the following selections on the **NetManager** "Local Configuration" panel:

Network Type:	. . . . .	Non-NetInfo network
Hostname:	. . . . .	*your hostname*
NetInfo and Configuration Server:	. .	**not** checked
NIS Domain Name:	. . . . .	None
Address:	. . . . .	*ethernet-network address if any*
Router:	. . . . .	None
Time Standard:	. . . . .	Disabled
Netmask:	. . . . .	Default
Broadcast Address:	. . . . .	Default

Since you don't have a local Ethernet, you shouldn't connect either of the Ethernet interfaces on the NeXT system to a working network (or even a pair of 50Ω terminators on a BNC "T" connector.

The important thing to remember when trying to configure your NeXT for SLIP operation is that the NetManager application will be of little to no help at all. In fact, if you attempt to use NetManager to configure network addresses, subnet masks, etc, you will become hopelessly confused and lost. Save yourself! Don't do it! Just say **no** to NetManager!

If you *do* have a local ethernet, then you need to configure the Ethernet interface's address with **NetManager** as appropriate.

Selecting IP addresses

The IP address, subnet mask, and remote IP address that you choose for a SLIP interface must be appropriate for the environment that you are using. The SLIP Server has an address which it is expecting you to use, since it is attempting to route traffic to you. Similarly, you must know the address of the SLIP server since you are attempting to route traffic to it and other points beyond. These addresses are normally provided to you by the administrators of the SLIP server.

Here's a typical example:

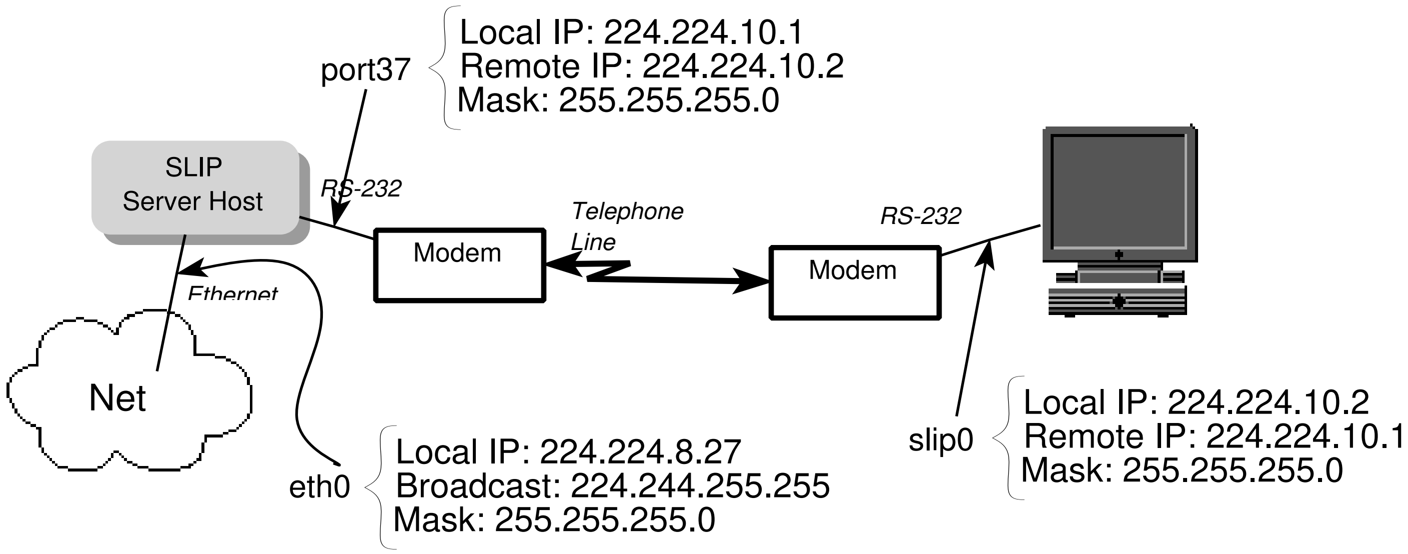

Figure 2

*(Please note that the addresses in this example have been invented, and should **not** be used at your site. If you require IP network numbers assigned, you should apply for one and not inadvertently use someone else's. This is considered very rude, and is **not** a good way to make friends in the Internet community.)*

In this example, the SLIP server is connected to an ethernet network, and has an address on that ethernet of 224.224.8.27. The class B network (224.224.0.0) is subnetted, with 8 bits of subnet number (determined by the subnet mask; note that it is identical on **all** interfaces on the 224.244.0.0 network). On one of its dial-up SLIP ports, the address is configured on subnet 10; the local address on the SLIP server is 224.224.10.1, while the remote address is 224.224.10.2. Note that the configuration on the NeXT SLIP client host is just the reverse.

Presumably, the SLIP server is advertising routing information for subnet 224.224.10.x on the Ethernet so that other hosts on the Ethernet, and perhaps on the Internet as a whole “know” how to reach hosts on the back end of the SLIP server. The dial-in clients will likely have default routes configured to point out the SLIP line to be able to transmit packets to other hosts on the network.

Note that the routing configuration has to be thought about and planned in both directions; it is usually not useful to be able to transmit packets, and note receive replies. The routing configuration for the “return” traffic to the dial-in SLIP host must usually be configured by the administrator of the SLIP server or the network manager. This is why it is **essential** that the correct addresses be chosen so that this all “works.”

Three different ways to use the DialUp-IP software

When considering the configuration of the DialUp-IP software and each of the SLIP interfaces, it is useful to consider some of the common ways to configure each of the SLIP interfaces as falling into one of three different models:

° Dial on-demand outgoing interface. This model is such that the interface is configured to automatically dial the modem when a packet is attempted to be transmitted on the interface. So, when you invoke a network application like 'telnet' or 'NewsGrazer' which needs to access a network resource on "the other end" of the SLIP interface, the modem is dialed.

After some (possibly infinite) period of inactivity, the line is dropped freeing the modem for other uses.

° Manually initiated outgoing interface. This is much like the case above, but manual intervention is required to cause the dialing daemon to choose a modem and dial. This might be useful if you normally use your modem for other purposes, like transmitting and receiving FAXs, and don't want it to be unavailable at unexpected times.

° Passive or server-like interface. This configuration has the NeXT act as a SLIP server for remote systems. The NeXT system has one or more modems attached to its serial ports for dial-in access. To configure access to a network interface (like **slip0**), it is necessary to generate a user account with a special shell: **/usr/dialupip/bin/dudisc_slip0**, for example, for the slip0 interface. When the remote system "logs in", the special shell reconfigures the serial line to be a SLIP interface and connects it to the named SLIP interface.

Note that it is still necessary to configure an address and other parameters in the **/usr/dialupip/config/config.slip** file. It is not necessary to add an entry in the **/usr/dialupip/config/diald.conf** file if the SLIP interface will not be use for dialing out.

Modem Scripting and Dialing on Demand

The DialUp-IP software has the capability of automatically establishing a link when packets are sent on an interface. When a SLIP interface is configured but idle, it is normally not "bound" to a particular serial hardware port. To cause the interface to become active, a binding is done between the network interface and a particular serial port. This can be accomplished in one of three ways:

• When a packet needs to be transmitted on a SLIP interface, and it has no hardware, but is configured for autodialing: a process is created to locate a free serial port/modem, and to dial the modem and log into the remote SLIP server.

• Similar to the case above, but where the interface is manually "brought up" rather than being brought up by sending a packet on the interface.

• When the NeXT is being used itself as a SLIP server; when someone logs in and runs the **`dudisc`** command to put the port into "SLIP mode".

In the first two cases, it is necessary to configure in the file `/usr/dialupip/config/diald.conf` for each SLIP interface (*e.g.,* `slip0, slip1`...) what serial ports, speeds and which dialing script is to be used to establish the connection. When the time comes that the link needs to be brought up, this file is consulted to begin the process of establishing a connection with the remote SLIP server.

It is not necessary to configure SLIP interfaces which are not dialed; the connection between the serial port login session and the SLIP interface is done by the **`dudisc`** command which is invoked as the login "shell" for that pseudo-user.

Software Installation

Now that we've talked about *what* and *why*, lets cover the ***how*** part. This section will cover the step-by-step actions to be followed to installed the DialUp-IP software on your NeXT computer system.

Installer application and package file

The DialUp-IP software is distributed in one of two ways; either on a floppy disk or as a UNIX **`tar`** archive image. The floppy distribution consists of a standard NeXT **`SLIP_v1.pkg`** directory which is intended to be used with the NeXT **/NextApps/Installer.app** program. The tar archive, when extracted,will create the same **.pkg** directory for the Installer application.

Invoke the Installer application (while being logged in a the UNIX super-user, **root**) on the package by opening the **`SLIP_v1.pkg`** directory which will launch the Installer application. The default location in the UNIX file system for installation is `/usr/dialupip` and should not be changed unless you have a *very* good reason. The DialUp-IP software assumes that the various files will be installed in this directory and will likely become horribly confused and malfunction if this is not the case.

If you need to reinstall the software, you can do so without disturbing and configuration files that you might have created since these are not part of the software distribution.

Description of installed files

Quite a number of files are installed; all of which are in the `/usr/dialupip` directory on NeXT system. What follows is a quick description of some of the more important files and what they are used for.

/usr/dialupip/bin/cslip_reloc: This is the actual loadable operating system kernel device driver. It is automatically loaded into your kernel by the **tcldiald** program at system boot time.

/usr/dialupip/bin/cslip_reloc10: Same as **cslip_reloc** above, but with 10 SLIP interfaces configured rather than the default of 2 interfaces.

/usr/dialupip/bin/tcldiald: This is the dialing daemon program that will select and dial a modem attached to your system when a SLIP link need to be brought up on demand. It also loads the kernel device driver at boot time. The **tcldiald** program is normally started at boot time from the **rc.slip** shell script.

/usr/dialupip/bin/duioctl: The duioctl program allows you to configure certain parameters of each SLIP interface (such as the MTU, if TCP header compression is enabled, etc), as well as enabling and disabling the dial-on-demand capability. It is also used to examine the state of each SLIP interface and to clear any error status which might be pending.

/usr/dialupip/Documentation.bshlf: this is a **Digital Librarian**™ bookshelf which contains documentation on the DialUp-IP software. This document as well as UNIX-style manual pages and other relevant information is indexed and available.

/usr/dialupip/man: A directory containing the actual documentation files, directories containing the UNIX manual pages, and other information (such as RFC's describing the SLIP protocol and TCP header compression).

/usr/dialupip/log: Various log files are created and written to in this directory which provide information about the running of the DialUp-IP software as well as trace information when that is enabled.

/usr/dialupip/config/diald.conf: This is one of the more important configuration files which configures the dial-on-demand SLIP interfaces. Information specified here includes the SLIP interface name, serial port and speed, system name, dialing script and telephone number. See the UNIX manual page **diald.conf(5)** or search for diald.conf in the Digital Librarian using the Documentation bookshelf for further information.

Note that this file is not installed since it contains system specific configuration information, but like other files in the **/usr/dialupip/config** directory, a sample is provided in the **/usr/dialupip/config/SAMPLES** directory to be modified for your site.

/usr/dialupip/config/rc.slip: This shell script is invoked at boot time to run the **tcldiald** program and configure each of the SLIP interfaces using both the standard UNIX **ifconfig** command as well as **duioctl**.

/usr/dialupip/config/config.slip: Note that all of the SLIP interfaces are configured here, and not only the ones to be used for dial-on-demand. This file contains the IP address, encapsulation and other configuration information for the SLIP interfaces on the system.

Note that a **rc.slip** file is not installed, but a sample version is available for modification in the **/usr/dialupip/config/SAMPLES** directory.

There are also a number of sample script files available in the **/usr/dialupip/config/SAMPLES** directory. Note that some have the suffix `.script` (which are used with the **diald** program), while others have the suffix of `.tcl` (which are used with the **tcldiald** program). Make sure that you base your scripts on the correct samples.

The sample scripting files are named according to a pattern (really! there is a method to this madness!) A series of files named dial-modemtype.tcl are present which are invoked to configure and dial a particular modem. Other files, named login-slipserver.tcl are invoked to possibly log in and put the remote system into SLIP mode. Finally, a file called hooks.tcl is present which contains utility and boilerplate information used by most dialing scripts.

The motivation for all of the different script files is to be able to reuse modem dialing and login scripts easily; they should not contain configuration sensisitive information.

If you are starting from scratch and have no existing scripts, the TCL based scripts used with **tcldiald** are recommended.

The /usr/spool/uucp/LCK

The software in this package that dials the modem on demand conforms to the *de facto* locking protocol that UUCP (and also kermit, and a number of other program) uses to arbitrate access to the serial devices on the system. Lock files are created in the `/usr/spool/uucp/LCK` directory, and this file must exist for the dialing function to work properly.

This directory is normally included in as part of the standard NeXT system software, but may have been deleted inadvertently.

Documentation access with Digital Librarian

The documentation for the DialUp-IP software may be distributed non-indexed to reduce the size of the software distribution.

To ensure that documentation is indexed, double-click on the icon for the file `/usr/dialupip/Documentation.bshlf` from the Workspace browser to invoke the Digital Librarian application. You can then double click on the Documentation icon to bring up an indexing panel; click on "Create Index" to begin the index creation. You will likely have to be logged in as "root" in order to create the index.

Configuration steps

To configure the DialUp-IP software, the following steps should be followed:

° If this is an initial installation of the software, you should copy all of the files in the **/usr/dialupip/config/SAMPLES** directory into the **/usr/dialupip/config** directory to serve as a starting point. This will copy sample configuration and scripting files which you can edit and customize. Note that *no* user modified configuration files in the **/usr/dialupip/config** directory will be modified or replaced by installing new versions of the software. The distribution package will have updated and new files shipped in the SAMPLES directory only. Thus, it is “safe” to install a new package on an existing system.

° **edit /usr/dialupip/config/config.slip:** Copy the sample version of this file from the /usr/dialupip/config/SAMPLES directory into /usr/dialupip/config, and edit the copy.

The sample **config.slip** file configures a single SLIP interface, **slip0**. The local, remote and netmask parameters for the SLIP interface are set in the file; you should change the sample values to those appropriate for your environment.

For each SLIP interface, a serial of shell variables are initialized with values. The first three, **SLIP*x*LOCAL**, **SLIP*x*REMOTE** and **SLIP*x*NETMASK** (where “***x***” should be replaced with a digit; 0 for interface slip0, for example) are used to specify the IP address information required for the interface. The **SLIP*x*CONFIG** variable carries options for the duioctl program which can specify other options for the interface.

By default, the interface configuration specified on the **SLIP*x*CONFIG** line sets the interface for the plain “SLIP” encapsulation. If you wish to use TCP Header compression, then the **SLIP*x*CONFIG** line should be changed to specify CSLIP rather than SLIP. If the SLIP server that you are using doesn’t support TCP header compression, you should not specify “CSLIP”. Also remember that unless you have a valid License Key file which enables the TCP Header Compressed version of SLIP, this line will have no effect. See the section *“License Keys”* for more information on License Keys and how they work.

If a particular interface should be used as the path for a default route, then the values of the **SLIP*x*DEFAULT** variable should be set to "**YES**". Only one interface should likely have a default route associated with it.

° **edit /etc/rc.local:** It is necessary to add a line near the end of the **/etc/rc.local** file on your system to invoke the **/usr/dialupip/config/rc.slip** shell script at boot time. You should put this line near the end of the file:

```
 sh /usr/dialupip/config/rc.slip         >/dev/console 2>&1
```

It is necessary to invoke the **rc.slip** file in order to configure the interfaces and cause the loadable kernel device driver to be installed.

° **edit /etc/syslog.conf:** This file needs to be modified to direct system logging messages related to the DialUp-IP software to a specific log file. Add this line to the **/etc/syslog.conf** file:

```
local3.info                             /usr/dialupip/log/syslog
```

A sample **syslog.conf** file is available in the **/usr/dialupip/config/SAMPLES** directory with the logging level set to "debug" rather than "info" for additional diagnostic logging messages.

° **/usr/dialupip/config/keyfile**: As shipped, no explicit License Key file, **/usr/dialupip/config/keyfile** is provided; the **rc.slip** script will automatically use the provided **/usr/dialupip/config/keyfile.slip** as the License Key instead. If you have another License Key that you wish to use, it should be installed as the **/usr/dialupip/config/keyfile** file. For instance, some versions of this software package are shipped with another keyfile, **keyfile.demo** which allows limited use of the TCP Header Compression feature. See the section on *"License Keys"* for further details.

° **edit /usr/dialupip/config/diald.conf:** The **diald.conf** file contains the information required to cause the NeXT computer to dial a modem and connect to a remote site in order to establish a SLIP connection. The connection establishment can either be initiated manually, or can be done automatically when a packet needs to be sent on a SLIP interface which is not currently connected.

A sample prototype **diald.conf** file is available in the **/usr/dialupip/config/SAMPLES** directory, and should be copied to **/usr/dialupip/config/diald.conf** and edited. The file consists of a number of entries, one entry per line. Lines which begin with a "#" character are considered comments, and are ignored. Let's examine a sample line in the **diald.conf** file:

```
slip0:annex.dom.ain:cufa#19200:slip-srv.script 555-1234:/usr/dialupip/log/annex@1:slipsrv.access
```

Each *entry* consists of a number of *fields*, each field is separated by ":" characters from the next. The first field in each entry is the name of the SLIP interface, in this case **slip0**; this is the same name used on the **ifconfig** and **duioctl** commands. Note that you are not free to choose the interface names; they are installed by the kernel device driver and are from the series **slip0, slip1, slip2**, etc.

The second field is the name of the remote SLIP server, and is commonly the doman name associated with the remote address. In this example, it is **annex.dom.ain**.

The third field is used to specify the name of the device and speed that the device should be set to. For example, to use `/dev/cufa` which is serial port A, with hardware flow control at 19,200 bits per second, you would specify `cufa#19200` in this field. Note that the /dev/ prefix is omitted. It is usually necessary to use the /dev/cua (or /dev/cufa) style device name since this UNIX device will open without carrier detect being present. This is necessary to be able to send dialing commands to the modem.

The fourth field is used to name the dialing script, and parameters passed to the dialing script, which will be invoked to dial the modem and connect to the remote SLIP server. This file name is relative to the `/usr/dialupip/config` directory if a complete path name is not supplied.

The parameter(s) specified are made available as arguments to the dialing script and are typically used to pass along telephone numbers, user names, passwords, etc. The `tcldiald` program itself doesn't attempt to interpret any of the parameters.

When using the `tcldiald` program, the file specified with a suffix of `.tcl` will first be used if present, otherwise the named file will be used. This is for compatiblity with the older dialing daemon, `diald` and allows the same `diald.conf` file to be used even though the script files are completely different.

The next field is used to name a transaction log for this particular host. This file is rewritten each time the SLIP link is to be brought up and can be used to help diagnose connection or scripting problems. This should be a completely specified pathname which refers to a file in the `/usr/dialupip/log` directory. You can append @1 to the file name to enable futher debugging and tracing the transaction file, including tracing of all TCL scripting commands which are invoked.

The final field in the entry refers to a file which specifies under which circumstances (times and protocols) will cause the SLIP connection to be brought up if it is not currently connected. It is also used to specify the period of inactivity required for before the line will be hung up after it has been dialed. The default value in the **slip-srv.access** file for the inactivity time is very, very large and the modem will stay connected once the link has been brought up. This file can be referenced from more than one line in the diald.conf file is desired.

See the UNIX manaual pages for `diad.conf`(5) and `diald-access`(5) for more details. Refer to the manual page `tcldiald`(8) for more information about the dialing daemon. Information on the TCL command extension language can be found in the Tcl(5) manual page, while the manual page for `tcldiald-script`(5) can detailed information about TCL commands specific to this pacakage for scripting and pattern matching.

Figure 3 depicts the relationship of the various file related to the dial on demand capability of the package. The motivation for the seemingly large number of files is to be able to re-use certain files, such a modem dialing scripts easily. It also isolates invarient information (how to dial a modem) from information which varies per site (such as which number to dial).

SLIP Dial-on-demand script files

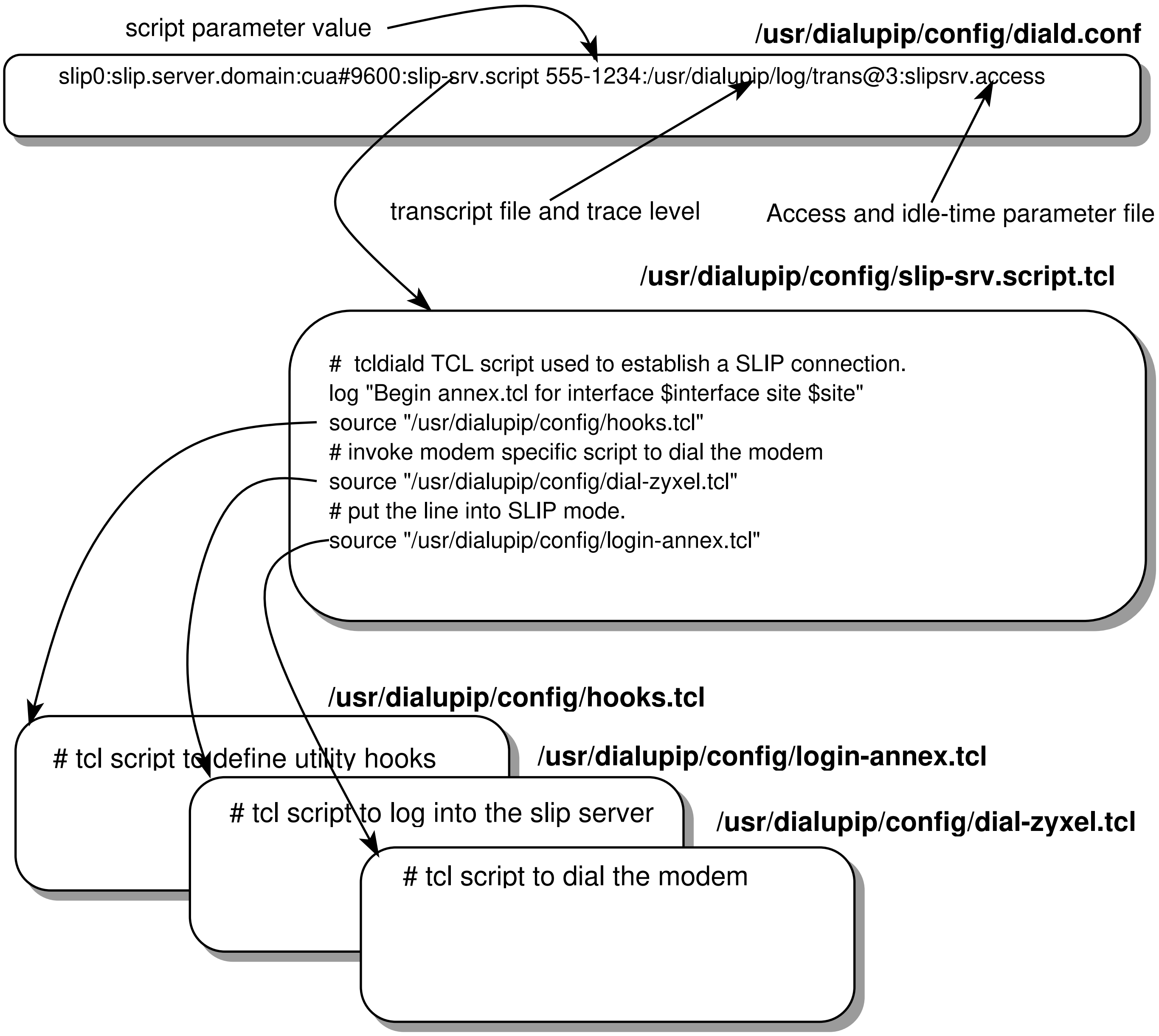

Figure 3

A Tour Through the Configuration Files

In this section, a complete set of configuration files is presented, along with annotated notes and (hopefully) helpfull commentary. The actual text of the file will be presented in the `Courier` font, while commentary will be in italics.

The **/etc/syslog.conf** file:

This file has one line added to it for logging purposes.

```
*.err;kern.debug;auth.notice                      /dev/console
kern.debug;daemon,auth.notice;*.err;mail.crit /usr/adm/messages
lpr.debug                                         /usr/adm/lpd-errs
mail.debug                        /usr/spool/mqueue/syslog
```

(This next line was added to capture log messages associated with the Dial-Up IP Software.)

```
local3.debug                      /usr/dialupip/log/syslog

*.alert;kern.err;daemon.err                       operator
*.alert                                           root

*.emerg                                           *
```

The **/usr/dialupip/config/config.slip** file:

This file contains the IP address and SLIP configuration information for **all** SLIP interfaces on the system:

The initial part of the file is just commentary to help describe how information is specified.

```
#  Specify configuration information for SLIP interfaces.
#  This file should
#  consist of sets of shell variable initializations for each interface.
# ------------------------------
#
# replace 'x' with a number for each SLIP interface in use.
#
# SLIPxLOCAL is the address of the SLIP interface on the local host.  This
# absolutely needs to be coordinated with the SLIP server or host that
# you'll be dialing into.
#
# SLIPxREMOTE is the address of the remote (SLIP server) end of the SLIP
# link.  This must be coordinated with the SLIP server or host that you'll
# be dialing into.
#
# SLIPxNETMASK
# This is the IP network mask of the subnet which is in use.
#
# SLIPxCONFIG
# duioctl options for interface
#
# SLIPxDEFAULT
# default route?  Should be set to YES or NO
# ------------------------------ slip0

SLIP0LOCAL=224.223.11.211
```

This first configuration directive sets the local IP address of the slip0 SLIP interface.

```
SLIP0REMOTE=224.223.11.111
```

The next directive sets the IP address of the remote system. If the remote address is unknown or varies each time its called, configure this the same as the local address.

`SLIP0NETMASK=255.255.255.0`

This configures the IP netmask of the local interface.

`SLIP0CONFIG=CSLIP`

This directive set the commands passed to the duioctl program to configure the mode of the SLIP interface. In this case, the interface will be used in Compressed SLIP mode. Usually, this directive will have no parameter passed. If you don't want this interface to be dialed on demand, specify DISABLECALL on this line too. This will cause the link to only be brought up and dialed when manually invoked by the duioctl program using the BRINGUP command.

`SLIP0DEFAULT=YES`

This will cause a default route to be installed using this interface. You cannot specify a default route here unless you have also specified a remote address above. Only one SLIP interface may have a value of YES for this parameter, and it may be that no interfaces have the default route to be installed using this interface.

`# ------------------------------ slip1`

The same sort of commands may be specified for each additional SLIP interface used on the system, for slip0 .. slip9. If you use more than two SLIP interface, you should be using the cslip_reloc10 kernel driver by rename it ti cslip_reloc.

A sample slip-srv.script.tcl script file. This is the base script invoked by tcldiald to bring up the link.

```
#  tcldiald TCL script used to establish a SLIP connection.
#
```

First, say hi to the folks at home...

```
log "Begin annex.tcl for interface $interface site $site"
```

Include some base-level support TCL code that most all dialing scripts will use. We'll take a look at the contents of this file in a bit.

```
#
# invoke 'hooks.tcl' script which contains default hooks and other
# setup stuff
#
source "/usr/dialupip/config/hooks.tcl"
```

The general plan now is to invoke two scripts; the first to configure and dial the modem. The second script is used to login to the SLIP server an turn on SLIP mode.

In this case, we're using a ZyXEL modem, so we invoke a script specific to that modem.

```
#
# invoke modem specific script to configure and dial the modem
#
source "/usr/dialupip/config/dial-zyxel.tcl"
```

The SLIP server we're dialing into is a Xylogics Annex terminal server. We've got a script to talk to it and turn on SLIP.

```
#
# and now invoke another script to actually log in and put the line
# into SLIP mode.
#
source "/usr/dialupip/config/login-annex.tcl"
```

This is the standard **hooks.tcl** file. You can modify as you see fit:

*First, get some interface statistics for logging purposes. **duioctl** is a TCL primitive that allows access to the same sorts of information as the **duioctl** program.*

```
#
set mtu    [duioctl SIOCGIFMTU]
set atimeo [duioctl SIOCGATIMEO]
set wtimeo [duioctl SIOCGWTIMEO]
set ipkts  [duioctl SIOCGIPKTS]
set opkts  [duioctl SIOCGOPKTS]

log "mtu $mtu atimeo $atimeo wtimeo $wtimeo ipkts $ipkts opkts $opkts"
```

*The main purpose of the stuff in this file is to define a number of TCL **hooks**, which are subroutines invoked by the tcldiald program as various events occur.*

*This hook, **event_linkdown** is invoked when the SLIP link is going down; either due to inactivity or loss of carrier from the modem. It is not invoked with any arguments.*

```
proc event_linkdown {} {
      global interface site
```

For lack of anything better, just log a message.

```
      log "Link DOWN event for $interface for $site"
      return "0"
}
```

*This **event_linkup** hook is invoked after the SLIP link is invoked and the dialing script returns normally. It's only parameter, **$why**, is a variable initialized to **packet** if the link was brougt up due to a packet being transmitted on the SLIP interface; or **bringup** if it was brought up manually.*

```
proc event_linkup {why} {
	global interface site
```

*The variables **$interface** and **$site** are initialized automatically when the interface is first brought up.*

```
	log "Link UP event ($why) for $interface for $site"
}

# note here how we only print a message at most every 20 minutes, no
# matter how often we are invoked
```

*The **event_uptime** hook is invoked periodically (currently, every minute) with a parameter that specifies how long the link has been up and running in minutes. This version only logs a message every 20 minutes. You might also do something like measure the number of packets transmitted and received on the interface during the last minute and compute an average.*

```
proc event_uptime {uptime} {
	global interface

	if {[expr "$uptime % 20 == 0"]} {
		log "SLIP interface $interface up $uptime minutes."
	}
}
```

The following two hooks are invoked when the state of the SLIP interface changes. They are invoked with the old value as well as what the new value will be.

These values are passed as decimal numbers. ***event_ifflags*** *is invoked when the interface flags (as seen by the ifconfig command) change. The* ***event_softflags*** *hook is invoked when SLIP specific software state changes (as seen by the duioctl slipx GSOFTFLAGS command).*

```
proc event_ifflags {oldflags newflags} {
      global interface site

      log "$interface/$site IF flags changed from $oldflags => $newflags"
}

proc event_softflags {oldflags newflags} {
      global interface site

      log "$interface/$site SOFT flags changed from $oldflags => $newflags"
}
```

A sample dialer file, in this case for the ZyXEL V.32/V.32bis/V.42/V.42bis modem.

```
#
# tcldiald TCL script used to configure and dial a ZyXEL modem.
# This script assumes that the zero'th argument is the phone number to be
# dialed.
parity zero
```

This command sets the parity of the transmitted characters to 8 bits, no parity. You can also specifiy EVEN, ODD, ZERO or ONE

```
# The phone number to dial is passed as the 0'th argument
set number [lindex $args 0]
```

Get the phone number to dial. In this case, this is the zero'th parameter specified in the diald.conf file after the name of the dialing file. There can be more than one parameter specified. ***$args*** *is a TCL list containing all of the arguments.*

```
log "Start of ZyXEL dialing script, dialing $number"
```

The ***log*** *command puts a helpful message in the syslog file (usually /usr/dialupip/log/syslog) to indicate that we've started up the dialing script.*

```
# flush any pending command
xmit {\r}
```

Send a carriage return character....

```
sleep 1
```

...and wait one second.

```
# get modems attention
set timeout 2
```

The ***timeout*** *TCL variable contains the number of seconds subsequent* ***expect*** *and* ***rexpect*** *commands will wait to match input strings from the modem. Here, we're setting it to two seconds as we expect the modem to respond very quickly to our attemts to get its attention.*

*The following **foreach** statement is an idiom; we don't expect it to run more than once, but by being in a **foreach** statement, we can easily exit the scope of the loop by using the **break** command. When we want to try something, and then try a few more times until we receive what we're expecting, this sort of method seems handy...*

```
foreach i {once} {
        xmit {AT\r}
```

Transmit the attention sequence to the modem...

```
        expect "{*OK\r*}" break timeout {}
```

*...and we're expecting to see an OK response from the modem. If so, then **break** out of the foreach loop; if the expect timesout, then do nothing.*

```
        xmit {AT\r}
```

And now we repeat the process a few more time..

```
        expect "{*OK\r*}" break timeout {}
        xmit {AT\r}
        expect "{*OK\r*}" break timeout {}
```

*And if if didn't answer by now, then something is seriously wrong. Abort the dialing attempt by using the **error** command, along with a helpful, explanitory message.*

```
        error "Could not get the modem's attention"
}

set timeout 5
```

Set the timeout up a little higher. Now we'll turn off the command echoing from the modem to reduce clutter in the trace file.

```
# turn off command echo
xmit {ATE0\r}
```

Wait for an OK reply. If we get it, do nothing. If this expect times out, abort the dialing attempt.

```
expect timeout {error "waiting for OK"} "*OK*"
```

Now we emark on a rather elaborate attempt to fetch the firmware version from the modem, just to log it out of curiosity. We send the modem an ***ATI*** *command and wait for the reply to come back.*

```
xmit {ATI\r}
```

Use the ***rexpect*** *command, which performs regular expression pattern matches. If this times out, don't worry about it; this isn't essential. Otherwise, we are trying to match a pattern as specified. In the case of the Neuron version of the ZyXEL modem, the modem will respond with a string like "FAX 1414 " with some number of trailing blanks. The data that matches the first pattern in ( ) groupingwill be stored in the $1 variable. We then use the* ***regsub*** *command to strip trailing blanks off of the version string. Sure seems like a whole lot of trouble....*

```
rexpect timeout {} \
        "(\[A-Za-z].*\[^\r\n]+) \r.*OK\r\n" {set version $1
                  # the regsub to strip trailing spaces
                  regsub {(.*[^ ])\ +$} $version {\1} version
                  log "Modem firmware version $version" }
```

Send a command to configure the modem with all of the right modes. See the modem manual for details of all this junk...

```
# configure modem with proper parameters
xmit {AT&K2&N0M0V1Q0X5&C1&D2&H3&J0&L0&M0&R1&S0N1\r}
```

Wait for a reply. Expect either "ERROR" which means we screwed up the parameters specified above; "OK" which is what we expect, or check for a timeout..

```
expect timeout {error "waiting for OK"} \
        "{*ERROR\r*}" {log "Modem returned ERROR"} \
        "{*OK*}" {log "set parameters"}
```

Now, we're ***finally*** *getting to the point of this script. Dial the phone with the number we picked up above.*

```
# dial the phone
log "Dialing $number"
```

```
xmit "ATDT$number\r"
```

*Since it might take a while to make the phone connection, have the remote ring a few times, and the modems handshake with each other, we set the **expect** timeout to 60 seconds.*

```
# wait for connect message
set timeout 60
```

Now, look for a variety of connect messages. Abort the dialing script on timeout, or if the modem returns "ERROR" or "BUSY". We customize the log message based on what speed and error correcting and compression we are using. If you want to insist on a particular speed or protocol, modify the expect and error abort if you don't get what you expect.

```
expect  timeout                        {error "connecting"} \
        "{*CONNECT */V.32 9600T/V42*}"  {log "Connected V.32/V.42"} \
        "{*CONNECT */V.32 9600T/MNP*}"  {log "Connected V.32/MNP"} \
        "{*CONNECT */V.32b 9600T/V42*}" {log "Connected V.32bis/V.42"} \
        "{*CONNECT */V.32b 9600T/MNP*}" {log "Connected V.32bis/MNP?"} \
        "{*CONNECT *2400/MNP*}"         {log "Connected $expect_match"} \
        "{*ERROR\r*}"                   {error "Modem returned ERROR"} \
        "{*BUSY\r*}"                    {error "Remote modem busy"}

# set expect timeout back to somethine "reasonable"
set timeout 20
```

The **/usr/dialupip/config/login-annex.tcl** file:

This file contains the TCL script to log into a Xylogics Annex terminal server and put it into SLIP mode. This version is pretty dumb, in that the Annex doesn't prompt for any sort of user name or password. Logic to handle this could be added if necessary..

```
#  TCL script for use with tcldiald.
#
#  Used to put a Xylogics Annex terminal server into SLIP mode.  This version
#  does not expect any login or password prompts.  If these are present with
#  you terminal server, you can pick up additional arguments from the $args
#  array and use them to specify user name and password.
#
#  $Id: login-annex.tcl,v 3.4 1992/01/13 01:33:11 louie Exp $
#
```

Log a friendly message as we begin

```
log "Begin annex login"
#
# set parity of transmitted data
#
```

As before, make sure we transmit 8 bits, no parity.

```
parity ZERO
#
#  For auto-baud nonsense.. sometimes its necessary to poke at it a couple of
#  times before it figures out what speed the modem is at.
#
```

We now begin the autobaud ritual. Set a relatively short timeout interval and send a carriage return to the terminal server.

```
set timeout 2
xmit {\r}
sleep 1
```

Wait a bit, then begin a few attempts to get the terminal server prompt

```
foreach i {once} {
        xmit {\r}
        #
        #  look for annex command prompt
        #
```

We're looking for an ***annex:*** *prompt from the terminal server. Once we see it, break out of the "loop". Else poke it again with another carriage* *return and look some more.*

```
        expect "*annex:*" {break} timeout {}

        xmit {\r}
        expect "*annex:*" {break} timeout {}
        xmit {\r}
        expect "*annex:*" {break} timeout {}
        xmit {\r}
        expect "*annex:*" {break} timeout {}
        error "Couldn't autobaud annex: prompt from annex terminal server"
}
set timeout 10
#
#  send command to put terminal server into SLIP mode
#
```

Now, actually tell the terminal server to put this port into SLIP mode.

```
xmit "slip\r"
log "Entering SLIP mode"
#
#  Use regular-expression based expect command to match addresses in message
#  that the annex server emits.
#
```

Look for the response to the “slip” command that we sent, and pick out the IP addresses from the message.

```
rexpect {Annex address is ([0-9]+\.[0-9]+\.[0-9]+\.[0-9]+).+Your address is
([0-9]+\.[0-9]+\.[0-9]+\.[0-9]+)}
#
#  Rather than just printing this message, you could ifconfig the interface,
#  for example, by using the exec TCL command:
#
#    exec /etc/ifconfig $interface $2 $1
```

Though we don’t do anything with the addresses here, log them for future use.

```
log "Annex address is $1, my address should be $2"
#
# wait a bit for server to switch to SLIP mode
#
sleep 1
return "Connected"
```

License Keys

The operation of various features of the DialUp-IP software is enabled and controlled though the use of a *License Key* file. The license key file has the ability to enable features such as SLIP, and Compressed SLIP as well as control the circumstances under which the software will operate (until a certain date; for an interval after the system has been booted), and which system it will operate on (by matching the system's HostID).

There are two License Key files provided with the free version of this software. One enables "plain vanilla" SLIP operation for an unlimited length of time with no expiration date. This is supplied in the file `/usr/dialupip/config/keyfile.slip.`

The other license key file enables "plain vanilla" SLIP as well as "Compressed" SLIP (using Van Jacobson's TCP header compression algorithm) until a specified date. This is the "try before you buy" license key, and is supplied in the `/usr/dialupip/config/keyfile.demo` file. Versions of software released after this initial version will have a different version of the "demo" key file which permits operation of the SLIP software for 60 minutes after the system has been booted.

The `/usr/dialupip/bin/tcldiald` program reads the license key file and enables certain functions both within the `tcldiald` program as well as the loadable kernel device driver. By default, the `tcldiald` program will load a license key from the file `/usr/dialupip/config/keyfile`; it is also possible to specify an alternate license key file on the command line invocation of the `tcldiald` program.

<u>It is vitally important that you do not modify the contents of the key files</u>. They must be preserved byte-for-byte exactly as they were shipped. If they are modified, they will cease to function as they will no longer pass an the integrity check which is performed. The Key Files are just plain ASCII, can can be viewed with editors or just "cat" to determine what capabilities the Key File in question enables.

Software Use

This section discusses some of the more common aspects of running the DialUp-IP software on your system, and some of the maintenance tasks and diagnostic techniques you might make use of.

`duioctl` maintenance program

The `duioctl` program is used to set and examine software state variables in the loadable kernel device driver, as well at to request certain operations be performed.

You will likely use the `duioctl` program in the `rc.slip` file to set the characteristics of each SLIP interface. The interface can be in three different encapsulation modes:

SLIP
CSLIP
AUTOCOMP

which sets "plain vanilla" SLIP, SLIP with TCP header compression or "automatic" mode which starts in uncompressed SLIP mode, but will switch to compressed SLIP if a compressed SLIP frame is received on the interface. It is not recommended that AUTOCOMP be used, as typically you need to coordinate the encapsulation with the MTU (see below). Also note that CSLIP mode (for TCP Header Compression) will not operate if your license key doesn't enable CSLIP.

You may also want to set the MTU (Maximum Transmission Unit) of the interface to a value different from the default (1006 bytes) when using CSLIP encapsulation.

For example, the typical configuration for a CSLIP interface would be set with this command in the `rc.slip` file:

```
/usr/dialupip/bin/duioctl slip0 CSLIP MTU 256
```

To enable or disable the dial-on-demand capability on a particular SLIP interface, you can use either of these two commands:

```
/usr/dialupip/bin/duioctl slip0 ENABLECALL
/usr/dialupip/bin/duioctl slip0 DISABLECALL
```

You should not enable calling on an interface which is not listed in the `/usr/dialupip/config/diald.conf` file, since the dialing daemon will not know how to select a modem and dial the call. Note that by default all SLIP interfaces have `ENABLECALL` turned on when their address is configured by the `ifconfig` command in `rc.slip` as specified in the `/usr/dialupip/config/config.slip` file.

For SLIP interfaces used for incoming calls (to be used in the SLIP server role) or for interface that you don't want to be dialed automatically even though they are configured in the `diald.conf` file, you should set `DISABLECALL` after configuring the interface address in the `rc.slip` file.

To cause a SLIP interface to be brought up (provided that you've configured it in your `diald.conf` file), you can use a command like:

```
/usr/dialupip/bin/duioctl slip0 ENABLECALL BRINGUP
```

The ENABLECALL is used to make sure that dialing is enabled, and the BRINGUP command causes the dialing daemon to initiate a call.

Differences from the prior release of SLIP and release notes.

This version differs from the previously released versions of SLIP (which was available on the various Internet FTP archive servers) in quite a number of ways. More importantly, it is functionally upward compatible with the previous versions and it still freely redistributable with the two supplied license keys. This means that the base-level SLIP function provided by the previous releases are still present in this version.

> *(Begin soapbox...)* I believe that it is important that the NeXT platform have a freely available SLIP implementation if it is to be take seriously by the the UNIX workstation community. We expect SLIP to be available on Sun, DEC and other platforms; why not NeXT? (The silly NeXT network API defined in the kernel is one reason, I suppose, which drives away developers and prevents otherwise available code from just dropping and and working as it does on other BSD based kernels.) *(End soapbox...)*

Future versions of this software will likely include a re-implemented kernel driver which will hopefully be less sensitive to NeXT operating system kernel versions and internal changes. Plus, it give me the opportunity to change and re-do some stuff that I've been wanting to clean up a bit. Necessarily, the user-level dialing daemon **tcldiald** will also be updated to accomodate the newer kernel driver. This should be of no consequence to users of the software as there are no plans nor need to change any of the dialing scripts.

In 920904, since January 1993:

> A number of typographic changes have been made in the documentation to reflect the creation of *TransSys, Inc.* which now develops and markets this software.

In 920904:

> • This version is the final version in a series which has tracked the testing and release of NextStep Release 3.0. There were two prior versions, released following NextStep 3.0 PR1 (pre-release 1) and 3.0 PR2.
>
> • There were some minor bug fixes here and there, with no major problems reported. One correction of consequence is a fix to the kernel driver to support changing the MTU (maximum transmission unit) size of

a SLIP interface to a value larger than the default of 1006 bytes.

• An added feature is now a traceback which is logged in the event of an error in any of the TCL scripts. This is helpful to diagnose just why that modem dialer script isn't working as you expected.

In 920506:

• Attention SLIP users! You are *very* much encouraged to use only the **tcldiald** dialing daemon and to move away from using the older **diald** program. There are changes in the works and new features being planned, none of which include the **diald** program. While this release still includes **diald**, don't count on it being included in subsequent versions.

• The kernel driver has been tweaked slightly to run under NextStep 3.0 pre-release 1. This was a relatively minor change which should not affect the operation of the kernel driver under release 2.1 at all. Obviously, extensive testing under 3.0 have not been done possible; but no problems have yet been observed.

• TCL error reporting has been enhanced by an error walkback to help diagnose and locate TCL script errors which may occur. Also, it is now possible to vary the tracing level from the TCL script. This is handy to turn off tracing once the connection has been established so that subsequent TCL hook invocations are not traced.

In 920407:

• Fixed bug in kernel driver which would only allow up to 9 network interfaces to be attached. This interacted badly with the cslip_reloc10 kernel driver which asked for 10, but only got one. It is now possible to allocate up to 98 interfaces, though the interface names get weird after the first 10.

• New kernel driver debug flags will dump a one line to syslog for each packet sent and received by the kernel SLIP driver. Turned on by the OPKT and IPKT flags. See the **duioctl slip*x* GDEBUG** command.

• Fixed bug in **login-annex.tcl** script which didn't escape the literal