The CLX 5.04 Installation and Administration Manual

Franta, DJ0ZY (franta@t-online.de) and Ben, DL6RAI (clx@dl6rai.muc.de) November 20, 2000

The complete reference for installing and maintaining a CLX node on your packet system.

Contents

1 Overview of the CLX System

1.1 What is CLX?

PacketCluster nodes have been around since around 1985. The original PacketCluster idea came from Dick Newell, AK1A, and was running under DOS. Circa in 1992 Dick stopped the development of the PacketCluster software for amateur radio. Many systems are still using this relatively old DOS software today.

CLX is a system which clones a PacketCluster node. To the outside user, commands and features are mostly identical, remote PacketCluster nodes, which can be networked, don't see the difference between a generic PacketCluster node and CLX. So it fits well into an established network of PacketCluster nodes.

1.2 CLX — Another PacketCluster clone based on Linux

CLX, short for "Cluster Software running under Linux" has been under development since February 1994. For many years before we were dreaming of improving Pavillion's PacketCluster software, which was developed in the mid-eighties. Now the time had come.

In order to add features to the cluster software it was necessary to look for a new operating system platform. Under DOS it was practically impossible to improve anything. The 640k memory limit makes things very difficult when it comes to writing a multi user application with up to 60 users or more, handling port interrupts for internal and external TNCs and database queries. So we came to the conclusion to use the Unix platform for this new project. The Linux Operating System seemed like an ideal playground for us.

Here are some of the features of CLX

- Full compatibility with the original `PCxx` protocol and from a users' point of view. It comes with multitasking and different priorities.
- The software allows loops and multiple connects without generating endless DX spots and announce messages. This is a feature to make the network more reliable.
- CLX uses modern programming concepts like shared memory for common data, lex & yacc for syntax checking, shared libraries to keep binaries small, remote procedure calls for inter process communication. It is strictly modularised with its own dedicated process for every major task.
- CLX can be extended, you can add your own programs and software.
- You may re-use the old PacketCluster database files (`*.ful` files). They can be imported into the new system without too much trouble.

- CLX is based on a data base system called PostgreSQL version 6.1. Currently, the CLX code consists of about 60,000 lines of C++ code.
- CLX's connectivity is very flexible. It supports the generic AX.25 drivers in the Linux kernel, uses WAMPES, NOS, JNOS or TNOS and plain TCP/IP via telnet. You can accommodate even the wildest network configurations including hook-up to the Internet via a dial-up line. Through the use of universal connect scripts almost everything is possible in CLX.

First on-air experiments started in late May (1994) under the callsign DB0PV-6, which now has become DB0CLX. It currently runs on Linux Version 2.0.25 with Alan Cox's and Jonathan Naylor's AX.25 kernel driver and a KISS connection to the local digipeater.

A current list of CLX users can be found in 7.3 (CLX User List).

1.3 Hardware requirements

CLX runs on PC 386-40 with 8 MB memory. Well, it creeps. It sure loves a little more CPU power. A typical production-level CLX system like DB0MDX has a Pentium-233 and 64MB RAM. At DB0CLX, which is rather a low-profile installation, we currently have a 486-100 with 32 MB RAM and 500 MB disk space.

For an average Linux installation you will need about 200 MB. CLX is about 5 MB after the installation but the data grows quickly. If you keep log files, these can easily fill up to 50-100 MB. So 500 MB is certainly no luxury.

For a connection to the the external world you can either use a KISS TNC on a serial port, WAMPES, AX.25 over Ethernet or even plain TCP/IP.

1.4 Availability

CLX is available by anonymous ftp from the following address:

ftp://ftp.funet.fi/pub/ham/unix/Linux/cluster/clx

This site is mirrored by many other sites world wide so you may pick up CLX from another place. Be aware, however, that `ftp.funet.fi` is the place where the most recent version of CLX usually gets uploaded first. Thanks for the ham-admins of `ftp.funet.fi` for the nice service they provide to the ham community and we hope that we will be able to continue using `ftp.funet.fi` as our main distribution site.

KB8EHT has offered to make CLX available in the US under the following address:

http://www.timsnet.com/pub/clx

2 How to install the CLX software on your system

The installation of the CLX software requires some knowledge of Linux and Unix in general. It is not an easy task.

Here's a checklist for you:

- I know how to use an editor (vi/emacs/joe)
- I know how to add new users
- I know about `tar`, `zcat` and `gzip`

- I know about TCP/IP, networking, `ping`
- I know about `syslogd`
- I know how to make a new Kernel
- I know about the `lilo` boot loader
- I know how to add commands to `/etc/rc.*` or `/sbin/init.d`
- I know about `crontab` entries
- I know about the dynamic linker `ld.so` and shared libraries

If you can say **yes** to all of the above, you are prepared to install CLX. If not, you better get yourself a book about Linux/Unix and play around with your system to become familiar. This may take some time (weeks, months).

This version of CLX is glibc-based. That means, no longer will it require you to make symbolic links and find a special version of libc.so.5 as it used to be in previous versions. However, CLX will no longer run on an old libc.5-based system.

For CLX to run, the following software packages must be installed on your system.

- Perl (most CLX tools are written in perl)
- Expect which requires Tcl (if you want to write your own connect scripts)
- Postgres (CLX data storage is based on Postgres)

From CLX version 5.00 Postgres is no longer part of the CLX distribution. We used to have Postgres along with CLX for many years but lately Linux distributions come with rather current Postgres packages and so the hassle to remove a pre-installed Postgres from a new Linux system has become bigger than the benefits of providing Postgres along with CLX. So the decision was made to leave it out and you have to install it for yourself.

You need at least Postgres version 6.5 tp run with CLX 5.00.

2.1 Users in /etc/passwd

CLX needs one new user in your `/etc/passwd`:

```
User            Group   Home Directory              UID
-----------------------------------------------------------
clx_us          users   /usr/local/clx              101
-----------------------------------------------------------
```

It is necessary to add the clx_us before you unpack the software, so the files will have correct ownership and permissions. The UID is chosen randomly, it does not really matter.

2.2 Unpacking the software

The CLX software comes in a single huge `.tgz` file. The file should be installed as follows:

```
# cd /
# tar xvzf clx_XXX.tgz
```

This will unpack the CLX software in the directory `/usr/local/clx`. You will need approximately 5 MB of disk space to install it.

You have now completed step one of the CLX installation! Fine!

2.3 A new Kernel

You need a Linux kernel of version 2.0.36 with the following drivers/components:

- System V IPC
- TCP/IP networking
- Amateur Radio AX.25 Level 2 (optional)
- Radio network interfaces (optional)
- Serial port KISS driver for AX.25 (optional)
- BPQ Ethernet driver for AX.25 (optional)

Earlier or later kernel versions may or may not work. Earlier versions required patches (called ax25-module14f or the like). We are currently using 2.0.36 at DB0CLX. The 2.0.36 kernel sources come with the AX.25 drivers in place, but they must be activated.

Currently I am not aware of how the 2.2.x kernels work with AX.25. There used to be some problems which may be fixed now.

If you got to compile a new kernel, please consult your Linux distribution manual. It is usually a sequency of "make config, make dep, make".

Congratulations! You have now finished the second step of the CLX installation procedure.

2.4 Testing the AX.25 driver

CLX does not require you to use the AX.25 kernel driver. If you wish to do so, you should now make sure everything works. If not you may skip this section and continue with the next one.

To check whether the newly created kernel driver is working we will make some tests. First you should get the most recent AX.25 utility package. This package contains programs which will allow you to make use of the kernel drivers. Without them, the drivers are just sitting in the kernel doing nothing.

The old version `ax25-utils-2.0.12c.tar.gz` dated December 28, 1996 is no longer supported and should not be used any more. The current utilities are in a file called `ax25-2.1.42a.tgz` available from *ftp://ftp.pspt.fi/pub/ham/packet/linux/ax25*.

Grab them, compile them and install them in the default directories using `make install`.

If you are using a 2.1 kernel, you could just go and use the new utilities without patching the kernel. There is some information regarding this subject in the AX25-HOWTO document, available, for instance, from *http://sunsite.unc.edu/mdw/HOWTO/AX25-HOWTO.html*.

Now we must configure ax25d. Go to the `/dev` directory and make a soft link from the `ttyS?` where your TNC is connected to `/dev/tnc`:

```
# cd /dev
# ln -s ttyS0 tnc
```

Switch your TNC into KISS mode. With my TNC2 clone running WA8DED firmware, I use the following command:

```
# echo -e "\015\015\033@K" > /dev/tnc
```

The TNC should signal the transition into KISS mode by flashing the CON and STA LEDs three times. After that you should observe that the CON LED keeps flickering very hectically. This is a good sign! If it doesn't, your TNC is probably not in KISS more or it behaves differently.

Now edit the file **/etc/ax25/axports**. You must specify your "outgoing" callsign for the different devices you have. Each TNC device gets one line in that configuration file. I have just one TNC connected so my file contains only a single active line:

```
# /etc/ax25/axports
#
# The format of this file is:
#
# name  callsign        speed   paclen  window  description
#
kiss    DL6RAI          9600    255     4       DB0AAB Link (438.300)
# end of /etc/ax25/axports
```

Now start **kissattach** to bind the TNC port to the kernel driver:

```
# /usr/sbin/kissattach /dev/tnc kiss
```

Note that "**kiss**" is the symbolic port name which is referred to in the **/etc/ax25/axports** file. You may give it any name.

kissattach should say something like: "**AX.25 port kiss bound to device ax0**". However, if you get the error message "**TIOCSETD: Invalid argument**", you forgot to add the Serial Port KISS driver for AX.25 to the kernel.

Now start **listen** and you should be able to monitor some traffic on the channel:

```
# /usr/bin/listen

Port kiss: AX25: DC1SLM->DB0AAB v DK1KMR* <RR R R3>

Port kiss: AX25: DB0AAB->DC1SLM v DK1KMR <I C S3 R6> pid=Text
0000  The quick brown fox

Port kiss: AX25: DB6MK-1->DG7DAH-1 v DB0AAB <RR C P R6>

Port kiss: AX25: DG7DAH-1->DB6MK-1 v DB0AAB* <RR R F R6>

Port kiss: AX25: DB6MK-1->DG7DAH-1 v DB0AAB <I C P S7 R6> pid=IP
IP: len 44 44.130.56.134->44.130.56.21 ihl 20 ttl 64 prot TCP
TCP: 1489->119 Seq xe86f491d SYN Wnd 31744 Data 4
0000  ....

Port kiss: AX25: DG7DAH-1->DB6MK-1 v DB0AAB* <REJ R F R6>
^C
```

Interrupt it with ctrl-C.

We can now try to make a connect with the `call` program. Before, I had to adjust my TX delay parameter slightly to make the program work. At DB0AAB only very short TXDs are accepted, otherwise one will receive a message saying: "TX Delay too long". Adjusting the TX delay is done with the `kissparms` program.

```
# /usr/sbin/kissparms -p kiss -t 150
```

Then give it a try:

```
# /usr/bin/call -r kiss db0aab

GW4PTS AX.25 Connect v1.11
Trying...
```

You should see your PTT LED go on and off and after a while you should succeed to connect:

```
*** Connected to db0aab
Rawmode
PC/FlexNet V3.3e - Muenchen/Fachhochschule - PR56/RS36 - 9600Bd FSK
=>
```

Congratulations! You have now mastered step three of the CLX installation! After having disconnected, you may kill the `kissattach` program still running. We don't need it at this time.

2.5 Configuring TCP/IP

CLX needs TCP/IP to communicate. Theoretically, it would be possible to have several modules of CLX running on different machines as they are all using the same mechanism.

All what it needed now is that you can reach yourself by TCP/IP. This is probably working already. Try the following command:

```
# ping localhost
```

If you receive the following, everything is in order and you can continue with the next step. Use ctrl-C to abort ping.

```
PING localhost (127.0.0.1): 56 data bytes
64 bytes from 127.0.0.1: icmp_seq=0 ttl=255 time=1 ms
64 bytes from 127.0.0.1: icmp_seq=1 ttl=255 time=1 ms
^C
--- localhost ping statistics ---
2 packets transmitted, 2 packets received, 0% packet loss
round-trip min/avg/max = 1/1/1 ms
```

However, if you get the following answer, something is not yet set up correctly.

```
PING localhost (127.0.0.1): 56 data bytes
ping: sendto: Network is unreachable
ping: wrote localhost 64 chars, ret=-1
^C
```

In one of the rc files in `/etc` or `/etc/rc.d` (depends on your Linux installation) you must activate networking with the following commands:

```
ifconfig lo 127.0.0.1
route add localhost
```

See your Linux documentation for details. Having mastered this, you have now completed step four of the CLX installation.

2.6 Adding Shared Library Paths

With version 3.00 of the CLX software, we have introduced a shared library image in the `~/lib` directory. For shared libraries to work in general, it is necessary to enable the dynamic linker `ld.so` to find these images at runtime.

Generally under Linux there are several ways to get around the problem:

1. setting the `$LD_LIBRARY_PATH` environment variable
2. make symbolic links to the default directories `/lib` or `/usr/lib`
3. Add the paths to `/etc/ld.so.conf`
4. Use statically linked programs

Option 3 is the recommended way because this gives flexibility and should not be a problem. As root, edit the file `/etc/ld.so.conf`

```
# vi /etc/ld.so.conf
```

and add the following line

```
/usr/local/clx/lib
```

to the bottom of the list. Then save the file and inform `ld.so` to update its cache:

```
# ldconfig -v
```

It should now list the new directorie. You have now completed step five of the CLX installation.

By the way, you should remember that running this command after every update of the `libclxccl.so` is mandatory to make the programs see the new library.

2.7 Configuring Postgres

The Postgres package should be installed on your system but it needs some configuration before it can be used for CLX.

With the current SuSE 6.2 distribution I had to install two packages, namely `postgres` and `pg_ifa`. The second package contained interface programs to Postgres like `psql` which is used in many CLX scripts. Be sure to check this.

The next step is to initialize the database (if it has not been initialized any time before). Become user `postgres` and run `initdb`.

```
# su - postgres
$ initdb
$ ^D
#
```

Now, as root, start up the **postmaster** process. This is done using a startup script which should have been installed along with the **postgres** package.

```
# /sbin/init.d/postgres start
```

Be sure that a program named **postmaster** has started up now.

```
# ps auxw | grep postmaster
root       661  0.0  0.6  1208   416  ?  S    10:18   0:00 grep postmaster
postgres   235  0.0  1.5  4148  1000  ?  S    08:34   0:00 /usr/lib/pgsql/bin/postmaster -i
```

Here is an important detail: Check the options by which postmaster was started. The -o option allows passing options to the **postgres** program which is later called from within CLX. Here, the -F option states that fsync() is not called transaction. This parameter improves performance and makes your disks run more quiet but carries the risk of data loss in case of a power outage or system crash. You must decide if you want one or the other.

Also note that Postgres keeps its data in a directory pointed to by the $PGDATA environment variable. This directory should sit on a disk with a lot of space since this is the place where all the CLX data goes to.

Again, as user **postgres**, you must add user **clx_us** to the Postgres users. For this, become user **postgres** and use the program **createuser** to add a new user and allow it to create databases and new users. clx_us must have Postgres superuser privileges in order to perform some of the essential functions like importing/exporting tables from and to files.

```
# su - postgres
$ createuser
Enter name of user to add ---> clx_us
Enter user's postgres ID or RETURN to use unix user ID: 101 ->
Is user "clx_us" allowed to create databases (y/n) y
Is user "clx_us" allowed to add users? (y/n) y
createuser: clx_us was successfully added
$
```

2.8 Create CLX database and tables

Now we are ready to create CLX's database tables. Become user **clx_us** and run the **clx_db** script:

```
# su - clx_us
$ clx_db
```

This will bring up a number of messsages where you can see that tables are being created.

If you have any old data to reload (when you are coming from a previous installation of CLX), do so now using the **bup_db** script. If this is a new CLX installation you can skip the following step.

```
$ cd backup
$ cp ../../clx.old/backup/* .
$ bup_db -r
```

This restore process may take a while, depending on how many records are in your tables. If you have many users and lots of DX information, this will take even an hour or two. To monitor the process, check out the size of the database files in the directory `$PGDATA/base/clx_db`.

Create the index files on the database tables. This too might take quite a while:

```
$ clx_idx
```

Now we are done with everything and ready to start up CLX for the first time.

2.9 Starting CLX for the first time

We are now ready to start CLX for the first time. You might want to edit the default config file in `~/config/clx_par` but if you've followed the instructions so far, you should not need to make any changes. Leave the callsign string as it is now, the callsign will be `XX0XX`.

Now as `clx_us`, start CLX with the following command:

```
$ clx -u
```

You should now see CLX coming up for the first time.

```
Reading /usr/local/clx/config/clx_par.
Decoding callsign.

*** CLX System Parameters ***

callsign:   xx0xx
database:   clx_db
interface:  AX25

Checking directories and file permissions
Clearing shared memories.
Clx coming up...

clx_ctl  Shared Memory manager          ....up
int_com  Internal Communications Manager .. up
snd_ctl  Transmit Spooler               .. up
usr_ctl  User Administration            .. up
iu_com   Inter-User communication       .. up
icl_com  Inter-Node communication       .. up
usr_req  User Database Interface        .. up
mb_ctl   Mailbox Controller             .. up
udt_mng  User Data Table Manager        .. up
usc_mng  User Commands Manager          .. up
usr_dlg  User Dialog manager            .. up
bbs_if   BBS interface                  .. up
rm_disp  Received messages Dispatcher   .. up
rcv_ctl  Received messages Spooler      .. up
con_ctl  AX.25 interface                .. up

Clx is up.
$
```

When CLX is running, you will be returned to the **$** prompt. You can now log into CLX with the **term_usr** program. CLX should greet you with a message and then display the **clx >** prompt.

```
$ term_usr
Hi om, Tere is XX0XX! a Linux-PacketCluster node running CLX...
xx0xx-16 de xx0xx    4-Dec-1999 1145Z   clx >
```

Now you can try a few commands, like **SH/US**, **DX**, **SH/DX**, **SET/NAME** etc. Logout with "**BYE**".

Shutdown CLX with the following commmand:

```
$ clx -s
```

Congratulations! You have successfully installed CLX on your system. If you are now willing to run CLX on the air, continue with section 3 (Configuration) further down below!

2.10 Migrating from an older CLX installation

If you are migrating from an older CLX installation you will probably be interested in preserving some of the data you have acquired and some of the configuration work you have made up to that point. Here is a step-by-step description how you can save your previous work.

Basically the migration is easy as all user data is contained in either the database tables or the **~/box** directory. Additionally, your configuration is stored in **~/config/clx_par** and probably in **~/config/cluster_par**. So these are the things to backup.

1. Shutdown CLX and backup the user data

   ```
   $ cd
   $ clx -s
   $ bup_db -s
   ```

   The **bup_db** script will dump all database tables from a previous version and additionally store the mailbox messages in a **.tgz** file.

2. Login as root and rename CLX's home directory to **clx.old**. We will need to read some files from there later. Now unpack the CLX distribution from the archive.

   ```
   # cd /usr/local
   # mv clx clx.old
   # cd /
   # tar xvzf clx_XXX.tgz
   ```

3. If you want to replace the Postgres database system or if database structures within CLX have changed, you must perform the steps for initializing the CLX database as described in 2.6 (Configuring Postgres). You should consult the CLX release notes to decide if this step is necessary or not.

   When you are done, you should finish with the new **postmaster** process running and the CLX tables newly created, data read back in and index files created.

4. Go to the **~/config** directory and copy the old configuration files there. Edit the old one and add anything which you feel is necessary.

```
$ cd ~/config
$ mv clx_par clx_par.sample
$ cp ../../clx.old/config/cluster_par .
$ cp ../../clx.old/config/clx_par .
$ diff clx_par clx_par.sample
```

Also, you might be interested to copy the old `~/.profile` from the `~/../clx.old` directory, but watch out for any changes before you overwrite the default `~/.profile`.

This completes the migration steps. Easy, no?

2.11 Migrating from a Pavillion PacketCluster installation

The original Pavillion PacketCluster software and CLX have very little in common when it comes to programs. None of the original programs is working with CLX, database structures are quite different so generally installing CLX means starting from 0.

However, to make your and your users' lives a little easier we provide a migration tool for data.

- The `read_ak1a` program to import the files `DX.DAT`, `OPERNAM.DAT` and `WWV.DAT` into CLX tables. The `DX.DAT` at DB0BCC, for example, contains over 300,000 records of DX spots ranging back into the year 1992. This is quite a lot of useful DX information worth saving. The same is true for the `WWV.DAT` file, which holds records of solar activity over a number of years. `OPERNAM.DAT` contains names and location data of your users. Save them to save your users having to re-enter this information.

  Also `read_ak1a` is used to read QSL information data into the `qsl_mng` table and also general database files from Pavillion. In CLX, data in the QSL database table is special in such as way as the information is searchable in two ways, which makes it different from other `.ful` tables. Firstly, you can search by DX call but secondly, you can also search by QSL manager. This allows you to retrieve a list of DX stations for which a QSL manager is responsible.

  Details for using the `read_ak1a` utility are described in 5.9.3 (read-ak1a).

3 Description of the CLX Configuration Files

The CLX configuration files reside in the `~/config` directory. Here is where you have to specify your configuration and make changes.

3.1 Obtaining a callsign string

Why an encrypted callsign? Well, we have taken this approach to keep track of who is using our software. That is, if you want to use CLX on the air you need to have a **real** callsign. This is when you have to contact us. You can install the software, make tests etc. with the default callsign "`XXOXX`" which is encrypted as "`WpbANhEOuh;`".

But then, when you need a real callsign, write an email message to *clx@dl6rai.muc.de* sending a screen dump of the startup messages to us. This will be accepted as a proof that you have put some work into the project and are now to be rewarded with a callsign.

How do you do a screen dump? Many people seem to have difficulties with that. So here is what we want:

```
$ clx -u > proof.txt
```

Then mail this file to us, noting which callsign you want encrypted.

Email is also the best way to report bugs and problems and receive support. We use this means of communication because it is very fast and reliable and pretty common in the mean time. Writing letters and sending out disks is too much trouble for us.

As a good resource, you may also subscribe to the CLX mailing list. Thanks to Heikki Hannikainen, OH7LZB, for running the list server for us. If you wish to subscribe, simply send a message to *majordomo@lists.hes.iki.fi* with an empty subject. In the message body put the following line:

```
subscribe clx
```

From then on you will be on the list and receive any mail addressed to the mailing list address. Also you may send questions, proposals ideas to *clx@lists.hes.iki.fi*. If you ever wish to unsubscribe from that list, again send a message to *majordomo@lists.hes.iki.fi* with an empty subject line. The message body should contain one line like this:

```
unsubscribe clx
```

Please read the documentation thoroughly before you start to ask questions.

Thanks to Heikki for providing this service for free!

Another good resource is the World Wide Convers Channel 9000 which is often populated with many Clusse or CLX sysops. Clusse is a DOS-based software that was written by OH2LZB some years ago and is currently no longer maintained. If you have a WW-Convers node near you, join channel 9000, which is often entitled: `CLX/Clusse SysOPs`.

Arnold, PA4AB, makes available the CLX Home page at *http://clx.muc.de*. This page has links to all the interesting places, where to download the latest CLX, additional sources for databases, beta releases and an archive of the CLX Mailing List. Take a look!

3.2 CLX configuration files in ~/config

The following rules apply to both CLX configuration files `clx_par` and `cluster_par`:

- '`#`' is used as a comment character. Everything after a `#` in a line is treated as comment.
- Empty lines are OK.
- Every key word must start in column 1 and end with a colon.
- Parameters follow the key word. You can have any amount of white space (tabs, blanks) to separate the parameters.
- If you continue in a new line, the first column must contain white space.
- Maximum length for a configuration line (probably going over several physical lines) is 256 bytes.

3.3 CLX parameters in clx_par

The parameter file is read at the start of CLX. It contains important information about your system configuration like callsign, SSIDs, communication interfaces etc. Please note that syntax checking for this file is

almost non-existant. So if you make an error, expect strange things to happen. This hasn't been a problem in the past but please be careful.

There are two types of parameters, that we call variant and invariant. The variant type can be changed at runtime while invariant types cannot. For them to take effect, CLX must be shut down and restarted.

Parameters are:

- `call` (invariant), the encrypted callsign code for your CLX system. By default, you will have the demo callsign `xx0xx` as `WpbANhEOuh;` in this place. Once you have received a code for your personal callsign, you must put it in here. Beware, the `;` character above is not a comment! The callsign codes are always 11 characters long independent of the length of encoded callsign. If CLX cannot decode the callsign at startup (but if it sucessfully can decode the default callsign `xx0xx`) you probably have mis-spelled the code. It is best to use the cut&paste function to get the string into the configuration file.
- `sysop_info` (variant). This parameter allows you to specify sysop information (your callsign, phone number, E-Mail address or whatever) to display when users do a `SHOW/SYSOP` command.
- `ssid` (invariant), to specify an SSID to be used. From version 2.05 on callsign strings are encoded without SSIDs so you can add your own. The old encryption strings from pre-2.05 releases remain intact and may also be used with this feature.
- `prot` (invariant), the protocol to be used. At this time, only the PCxx protocol of the original PacketCluster software is available.
- `hops` (variant), which is used to specify hop information for broadcast type messages. The five different types correspond with the following PCxx protocol messages:

  ```
  dx      PC11
  ann     PC12
  user    PC16/PC17
  node    PC19/PC21
  wwv     PC23
  wcy     PC73
  ```

  and are specified like this:

  ```
  hops dx/99 ann/10 user/5 node/10 wwv/99 wcy/99
  ```

  The hops are applied to:

  1. locally generated DX spots
  2. DX spots coming from user-mode links

  They are definitely not applied to the spots you are receiving from other nodes. These spots are just decremented by one and if the counter is not zero, they will be forwarded as per the configuration. The only exception is the `hops_min` value which can be specified on a per link basis. See 3.4 (hops_min) for further details.

  When a negative number or zero is used here, this protocol message is suppressed by CLX and never transmitted. This way, some messages can be completely supressed, like limited protocol with PacketCluster.

  The `hops` parameter can also be used in the `~/config/cluster_par` file on a link by link base.

- `merge` (variant), to specify how many DX and WWV spots should be requested from the partner node to merge them into the database in case a link failure has occurred and spots were lost for a while. This will make CLX generate an appropriate PC25 (merge request) message. The amount to be requested is specified like this:

```
merge dx/20 wwv/5 wcy/5
```

- `ps_vers` (variant), to specify which PacketCluster version information is sent to the node being joined. Default is currently `5447`. The performance of CLX is not affected with this switch, so this is **not** to select a specific emulation mode.
- `db_name` (invariant), where the name of the CLX database must be specified (no need to change it)
- `db_port` (invariant), port where Postgres is listening, default is the Postgres port 5432
- `db_host` (invariant), specify, if Postgres is running on another host, default localhost
- `vacuum` (variant), to specify (in minutes) how often `vacuum` will be called from within CLX. If your machine is very slow or your database is very large, you might wish to set this to a big value so that it occurrs not so often. If you do not specify this parameter, `vacuum` is never called (which is bad). A problem is that when vacuum runs for a long time, the `clx_watchdog` (if you have it on) will time out and shutdown clx. See 5.6 (CLX Watchdog) for further details. You could also leave this parameter out of `clx_par` and call `vacuum` through cron as explained in 5.7 (Other regular tasks).
- `wampes` (invariant), to specify if WAMPES should be used as the default interface for outgoing calls. By default, CLX uses the AX.25 kernel driver by GW4PTS. This requires that WAMPES is installed on your computer. See 3.13 (WAMPES and CLX).
- `w_host` (invariant), to specify a WAMPES host to connect. If WAMPES is running on another host, you should specify it here. If WAMPES is running locally, you don't need this.
- `ax25` (invariant), can have the parameters `no` or `outgoing`. `no` prevents CLX from any activity on the AX.25 kernel socket. `outgoing` makes CLX use the kernel socket only for outgoing calls but leaves incoming calls to another application (in a standard Linux AX.25 configuration, a program called `ax25d` is used to sort this out).

  Note: If your Linux kernel has no AX.25 support, you **must** specify `no` here, otherwise `con_ctl` will hang at startup time.
- `callb` (invariant), to specify sub directories in the `~/callb` directory which contain callbook files. See 3.17.1 (Callbook CDROM data with CLX) for any details.
- `cba_drive` (variant), drive letter (DOSEMU) for accessing Flying Horse Callbook data.
- `dx_lim` (variant), in minutes; DX spots older than dx_lim minutes should be ignored. Default (if nothing is specified) is 70 minutes. If `dx_lim` is not specified at all, no limit exists. From CLX version 4.04, dx_lim also works into the future, so that spots newer than dx_lim are canceled. DX spots are not sent to local users. But they are still forwarded to other nodes, as we cannot locally decide what are old spots for other systems. Some sysops may argue that 15 minutes is enough, but when you are at the edge of the Packet network, you may still be happy receiving 30 minutes old spots.
- `wwv_lim` (variant), in minutes; analogous to `dx_lim` you can specify after which time WWV spots should be ignored. The default is 720 minutes (12 hours). If `wwv_lim` is not specified at all, no limit exists.
- `wcy_lim` (variant), in minutes; analogous to `wwv_lim` you can specify after which time WCY spots should be ignored. The default is 720 minutes (12 hours). If `wcy_lim` is not specified at all, no limit exists.
- `wwv_auth` (variant): A callsign which will be the assigned "authority" for WWV spots. If several spots arrive for a specific 3-hour time frame, only one such WWV message is saved and broadcasted to the local users.

The reason here is that with nowaday's network situation, WWV spots tend to show up three- to fourfold at 18 minutes past the hour regularly. When you do a **SHOW/WWV** you will find four spots for the last period and only one for the next to last period.

If **wwv_auth** is given, the spot from this callsign is selected, else the first WWV spot received for this time frame is selected.

- **dx_comm** (invariant): Setting this parameter to **no** means that the comment field in DX spots is ignored for duplicate DX spot checking. This comes in handy for European installations where often spots including specific national characters tended to be repeated over and again. Setting the parameter to **no** allows suppressing these spots with a slight chance that some additional spots may be lost too.

  Setting the parameter to yes or not specifying **dx_comm** at all makes CLX include the comment field in the dupe checking.

- **shm_dxel** (invariant): This config parameter allows setting the search depth for checking duplicate DX spots. The default depth is 500 spots, this parameter can be increased up to 99999 spots. An appropriate shared memory block will be reserved for this.

- **shm_annel** (invariant): This config parameter allows setting the search depth for checking duplicate announce spots. The default depth is 400 spots, this parameter can be increased up to 99999 spots. The size is currently 214 bytes per record.

- **pw** (variant), to specify a password for the **set/priv** command. The password must be between 8 and 255 long. In order for your users to be able to gain CLX admin status, they will need the password and also a private copy of the **get_pwd** program. See also 1 (Achieving Admin Status).

- **baycom_pw** (variant), to specify a password for the **pw** command. The password must be 5 to 255 characters long. The first character of the password is position 1 (not 0 like some C programmers might suppose). See also 2 (Achieving Admin Status).

- **filter** (variant) allows designing DX filters for users to specify. This command allows to set up a line of notch filters for the users to chose from. The filters have numbers from 0..31 and may be cascaded by the user. The users' filter setup is permanently stored in the users' us_data record.

  Filters in **clx_par** may be specified as follows:

  - by a frequency range
  - by a frequency limit
  - by mode
  - by an entry from ~/config/ar_band.cd

  The syntax is as follows:

  ```
  (filter_number>name)specification
  ```

  The maximum length for a filter name is 12 characters. A maximum of 32 filters may be defined by the administrator. Frequencies have to be stated in kHz. Using a "/" in the specification is a logical **and**. **"20"/cw** means take the definition of "20" from ar_band.cd and use only the "cw" band. The double quotes are necessary to let the parser know that "20" is to be treated as a string and not as the number 20.

  Here are a few examples:

  ```
  (1>VHF)144000-
  (2>HF)3500-30000
  (3>TOP)1800-2000
  ```

```
(4>SIX)50000-52000
(5>CW)/cw
(6>20CW)"20"/cw
```

A filter may also consist of several specifications like this:

```
(7>WARC)"30"
(7>WARC)"17"
(7>WARC)"12"
```

The WARC filter now contains all three WARC bands.

The most important thing about filters is that you must understand that they are used to **reject** specific DX spots. An example: Specifying VHF, WARC, SSB and RTTY filters means, you will only get CW spots on the classic HF bands 160, 80, 40, 20, 15 and 10. This is probably what users want when the are in the contest. Or - when the CW filter is set, you will get anything but CW spots. Specifying SSB and VHF you will only receive HF CW and RTTY spots.

- `mk_fwd` (variant), allows specifying which addresses should be bulletin addresses. Normally ALL and international variations like ALLE, TODOS, TOUS, TUTTI are placed here.

- `waz_ddd` (variant), allows the sysop to specify a list of up to 40 WAZ zones which are to be treated as "DX". DX spots originating from these zones may be filtered out by users using the SET/NODXDEDX command. Also, these spots are not forwarded to other nodes which are specially marked in the `~/config/cluster_par` file. See there for more details.

  Please keep in mind that using this functionality, some database traffic is being generated as each spot must be analysed and the WAZ zone of the author must be fount out.

  Also keep in mind that this feature works only for outgoing (ie. transmitted) spots at your end. It does not filter incoming (received) spots. So if you are receiving those spots and don't want to **receive** them, ask your internet link partner to set up a filter for you that prevents him from sending unwanted spots to you.

- `syslog_lev` (variant) allows specifying a default syslog level for all CLX programs. The default is now defined as 6 (all except DEBUG), it used to be 7 previously. One can use any value between 0 (EMERG) and 7 (DEBUG).

  When a program is suspicious, you can dynamically change the debug level with the -l switch. For example

  ```
  $ icl_com -l 7
  ```

  would turn on the DEBUG level for the program `icl_com`.

- `maillimit` controls the maximum age (in days) a mail message can have before it will be deleted by `db_maint`. If nothing is specified, the mail limit is 90 days be default. This is used with `db_maint`'s `6` option. See 5.10.4 (db_maint) for further details.

- `loglimit` (variant) specifies for how long log records are kept in the database before they get deleted by `db_maint`. The default value is 20 days. This is used with `db_maint`'s `6` option. See 5.10.4 (db_maint) for further details.

- `uslimit` (variant) specifies for how long user records are kept in the database before they get deleted by `db_maint`. The default value is 120 days. Entries with special rights (us_perm != 0) like digipeater call signs or sysops are never removed automatically. This is used with `db_maint`'s `5a` option. See 5.10.4 (db_maint) for further details.

- `dxlimit` (variant) specifies for how long DX records are kept in the database before they get deleted by `db_maint`. The default value is 100 days. This is used with `db_maint`'s `1e` option. See 5.10.4 (db_maint) for further details.
- `annlimit` (variant) specifies for how long announcements are kept in the database before they get deleted by `db_maint`. The default value is 100 days. This is used with `db_maint`'s `1g` option. See 5.10.4 (db_maint) for further details.
- `qslfile` (variant) specifies the file name to be used with `db_maint` for reading in QSL information into the QSL database table. This is used with `db_maint`'s `7c` option. See 5.10.4 (db_maint) for further details.
- `batchcommands` (variant) specifies db_maint command options to be executed when `db_maint` is run with the "batch" argument. This is a handy way to automate database maintenance jobs via cron. See 5.10.4 (db_maint) for further details.
- `bbs_lst` (invariant) is the list of callsigns to be treated as Packet BBS systems for personal mail forwarding. The callsigns must be separated by white space. Mailbox systems are expected to actively connect CLX to initiate the forwarding. They are then greeted with the typical BBS forwarding prompt. CLX itself does not start up a connection to a PBBS system. CLX currently understands the W0RLI forwarding protocol only.

  As a side node, you must be sure of the callsign which is used by the BBS system for outgoing calls. For example, the Baycom Mailbox DB0AAB-8 uses the callsigns DB0AAB-4 and DB0AAB-5 (alternatively) for its forwarding traffic. These are the callsigns that must be stated with the `bbs_lst` parameter.
- `qsl_rr` (variant) is used to specify a remote database for QSL requests (`SHOW/QSL` and `SHOW/MANAGER`) when the result is not found in the local database. The syntax is `qsl_rr: <node>/<dbq>/<dbm>`. The fields `dbq` and `dbm` may be left empty as the default is `QSL` and `MANAGER`. So it is enough to specify the remote node like this: `qsl_rr: <node>`. A remote database request (PC44) will be generated and sent to the remote node (which can be a Pavillion PacketCluster systen or CLX).
- `language` (invariant) specifies the default language to be used on this system. At clx startup time, appropriate symbolic links will be created in the config and help directories.
- `clust_loc` (variant), a list of nodes to be treated as local nodes for announcements.

Here is a simple `clx_par` file:

```
call:           WpbANhEOuh;     # Encrypted callsign
db_name:        clx_db          # name of the clx database
prot:           pc              # protocol type (pc, clx or aut)
hops:           dx/99   ann/10
                user/5  node/10
ssid:           8               # SSID to be used
mk_fwd:         all alle dx dxnews
dx_lim:         60              # no DX spots older than 60 minutes
wwv_lim:        180             # no WWV spots older than 3 hours
wcy_lim:        60              # no WCY spots older than 1 hour
waz_ddd:        3 4 5 25        # ignode spots from these WAZ areas
bbs_lst:        db0aab-4 db0aab-5 # BBS systems for mail forwarding
qsl_rr:         db0sdx/qsl      # remote QSL database fallback
language:       german          # default language used on this system
```

3.4 Cluster network configuration in cluster_par

The file `~/config/cluster_par` contains information on which callsigns are nodes, which ones have to be actively connected and which ones build up the connect themselves. Additionally, the file contains routing information and of what type of node the system is. Moreover, there is a parameter to denote a link either "active" or "passive".

This file is similar to the `NODES.DAT` of the traditional PacketCluster software but it contains a little more information.

The format of this file has changed dramatically in version 4.00 of the CLX software. The name was changed too from `~/config/cluster` to `~/config/cluster_par`. We tried to make it a bit simpler and get away from the hard-readable link characteristic flags. We hope you appreciate that.

These are the syntax rules for this file:

- Empty lines are ignored.
- You can specify comment lines with "`#`".
- Data fields may be separated by any number of blanks and/or tabs. Use them to make the configuration file look pretty.
- Every node has its own `SECTION:` paragraph. A section ends with a new section or at the end of the file.

There are a number of parameters for every single connection described below.

-
- `section` Starts a new section and specifies the callsign of the link partner on the other side.
- `conn_act` This parameter specifies, if the connection is `active` or `passive`. For a discussion of active and passive, please see below.
- `conn_call` If the link is an outgoing connection, this optional parameter specifies, which callsign is going to be used. By default, the default callsign is used. If it is an incoming link, you may specify, under which callsign the remote node connects. This is like an alias callsign.
- `conn_int` This option is used to specify the type of connection. Four types are available:

```
a      ax25 kernel socket connection
w      WAMPES port
x      script from ~/exec/connect
i      incoming connection
```

- `conn_lock` Specifies if the connection is locked - i.e. disabled. It may be temporarily unlocked using the `clx_adm` program.
- `conn_path`

  1. Connection type `a` or `w`: One or several alternate connect paths may be specified here. Simply specify the callsigns of the digipeaters to be connected. With `%`-characters added to the call, the timeout period is specified:

```
   % = 1 minute
  %% = 2 minutes
 %%% = 4 minutes
%%%% = 8 minutes
```

Default time out is 1 minute. The time out for the last hop (to the final destination) can be specified on a new line.

Additionally, the connect string to be expected must be specified.

```
1 = "*** connected to"
2 = "> Connected to"
3 = "} Connected to"
```

Or you can specify it literally by adding it in quotation marks: `db0fsg%%"#LINKED_TO"` hererby replacing spaces with the underscore character.

Here is an example:

```
db0bcc: db0uni%2 db0fsg%1
        %%
```

DB0BCC will be connected via DB0UNI and DB0FSG. The timeout for the connection to DB0UNI is 2 minutes, and the string "> Connected to" must be received within this period. In the next step, DB0FSG must be reached within 1 minute waiting for the string "*** connected to". Finally the hop to DB0BCC must be reached within two minutes. The trigger for the final destination is the `PC18` frame received or, in user mode, any string.

2. Connection type `x`: Name of the program or script to be called including up to 10 tab- or space delimited parameters.

Alternate routes must be separated with the "|" character.

- `conn_ping` This parameter specifies if the connection is being checked at regular intervals. Acceptable values are `yes` and `no`.

  If the protocol type is `pc` then a `PC51` telegram is generated. In user mode (protocol type `u`), the CLX software only sends the string `ping 1` and expects an answer (any string) from the other side.

- `conn_port` This command is for AX.25 outgoing connections only. It defines which port will be used for the outgoing call. State the symbolic name of the port from `/etc/ax25/axports` here.

- `conn_prot` Protocol type. CLX knows two:

  ```
  pc      The PCxx protocol known from Pavillion Software
  u       the user mode
  ```

  In user mode this mode, CLX makes a connection like a normal PacketCluster user, and reads the DX spots which are received in the traditional user format:

  ```
  DX de DL1SV:      7012.0  PJ5AA        listening zero beat            0426Z
  ```

  This extension makes it possible to establish link connections without the link partner having to treat your callsign specially (i.e. putting your callsign into his `NODES.DAT` file). If you wish to send spots to your link partner, you must ask him to configure your side as a node.

  It is no longer possible to transmit any spots as it used to be in pre-4.01 versions. Too much trouble with badly configured setups have led to this decision. In fact, a user mode link is always defined as a non-clx passive link connection ignoring whatever the configuration file says.

  The version reported by CLX for user mode links is 0000 so do not worry when a PC19 frame is recevied looking like this:

  ```
  PC19^0^R3ARES^0^0000^H2^
  ```

  A user mode link is reported here with R3ARES.

- `conn_type` With this parameter, you specify if the partner node is a CLX node or not. Available types are:

```
clx          A CLX node
non-clx      not a CLX node
```

CLX can have one active and multiple passive connections into any isolated network. In Pavillion terminology, you can replace your existing link connections with active connections and additionally introduce additional passive backup links for any case. The difference is that on a passive link, CLX does not transmit broadcast type information (DX spots, announcements) to non-CLX systems and so no duplicate spots are being generated.

For a list of broadcast type `PCxx` messages see 3.3 (Broadcast type PCxx telegrams). Be careful when designing your network connections here because when you are doing it wrong, you will see the good old cluster loop in full swing!

Look at the following matrix to understand which messages are relayed by CLX if the partner node is a CLX node or non-CLX system and if the link is active or passive:

		outgoing partner is a..			
		A-CLX	P-CLX	A-PCL	P-PCL
Incoming message is from a..	A-CLX	ABC	A	BCE	-
	P-CLX	A	A	-	-
	A-PCL	ABC	A	BCE	-
	P-PCL	A	A	-	-

```
Legend:

A-CLX = Active CLX node
P-CLX = Passive CLX node
A-PCL = Active PacketCluster node
P-PCL = Passive PacketCluster node

    A = DX, Announce and WWV (PC11, PC12 and PC23), no duplicates
    B = other broadcast-type messages besides A:
            User add/delete (PC16/PC17)
            Node add/delete (PC19/PC21)
            here status (PC24)
            User info (PC41)
            Local User Count (PC50)
    C = directed messages:
            Mail forward (PC28-33)
            Remote Database requests (PC34-36)
            Talk messages (PC10)
            Remote commands (PC34-36)
            File forwarding (PC40)
            Remote Database requests (PC44-48)
            Ping (PC51)
```

```
D = Initialisation commands (PC18, PC20 and PC22)
E = type-A messages previously received from a passive link
    connection and now being again received from an active link.
    Generally, PCL nodes do not receive spots from passive links,
    but these spots can be safely delivered to them.


Locally generated messages (DX spots entered at the local node)
are treated like if they came from an A-CLX node. D-type messages
are exchanged with all partners but never passed on to others.
```

- `hops` (variant), see description above in 3.3 (Hops).
- `hops_min` (variant), allows specifying a minimum hop counter value for frames going to that link. This parameter is the only one which affects foreign frames (i.e. not self-produced frames). The hop count can only be increased not decreased. The format is identical to the `hops` parameter. However, `hops_min` is specified on a per link basis (in `cluster_par` while `hops` is a general parameter (in `clx_par`).
- `init_wait` Specifies the number of seconds CLX waits after the connection is established, before sending the PC18 initialisation frame.
- `mail_fwd` Enable or disable mail forwarding to a specific node.
- `merge` Number of DX and WWV spots requested for merging when re-connecting this partner.
- `ping_to` This variable decides the link check mode for the connection. There are three cases:

  If it is not specified, the link check to the remote system is based on either data or a ping answer received from the remote system.

  If `ping_to` is set to 0, a ping is sent every 300 seconds. Also, every 300 seconds a check is made if the previous ping was answered within 300 seconds or if any other message was received from the other end - else the link will be disconnected.

  If `ping_to` is set to a value greater 0, this value will be taken as the maximum allowed response time (in seconds) for a ping frame. A ping is sent 300 seconds and at the same time a check is made if the `ping_to` timer has been exceeded.

  `conn_ping` overrides these settings. If `conn_ping` is off, no link checking is active.

  To resume: there are two modes of link checking available. One that is triggered by normal PC telegrams (any data activity on the link) and one which takes care of the ping time only (how much time does it take for the answer to arrive). Both modes can be used at the same time to reliably check the link connection. With the first mode you would make sure that some activity still exists, with the second you could automatically cut the link and restart it when the response time has become too large.
- `waz_ddd` A list of WAZ zone entries treated as DX zones. Spots originating from there (probably transported by Internet connections) are not forwarded to this partner. If the `waz_ddd:` tag is not speciefied, the default from the `clx_par` file is taken.

3.5 System messages in ~/config/adv_txt.*language*

This is the system messages file which may be used in its original form or it can be adapted to your local language. Its function is similar to `MESSAGES.DAT` of the original PacketCluster software.

The maximum length of the strings contained in every message is 512 characters. This may include \n and other printf-style symbols. You must put this into a single line, however, the message may be structured with additional \n's so the user will receive several lines in a single message.

The following macros are available:

```
\call           = User's callsign
\ccntr          = Number of links currently established
\fulldate       = Full Date
\hcall          = User's callsign with here/nohere status showing
\n              = Newline
\ncntr          = Number of nodes reported in the network
\nodecall       = Your CLX callsign
\ring           = Bell character (ctrl-G)
\shrtdate       = Short Date
\since          = Time when CLX was started
\t              = Tabulator
\time           = Current time
\ualm           = The current alarm string (if any)
\ualv           = The current alive character
\uann           = Show if user has ANNOUNCEMENTS ON or OFF
\uansi          = Show if user has ANSI on
\ubeep          = Show if user has BEEP turned on
\uchs           = The character set selected
\ucntr          = Number of users logged in locally
\udxd           = Show if user has DXDEDX activated
\udxf           = list of filters currently active
\udx            = Show if user has DX spots on
\uex            = The current exit string
\ulog           = Show if user has Login/Logout messages on
\umax           = Maximum number of users logged in since last start
\uptime         = Uptime in days, hours, minutes and seconds - format
                  string in adv_txt message 011.
\version        = CLX version number
```

Some message numbers have flags. These have the following meaning:

```
*       This message only goes to the error log. It is always
        taken from adv_txt so there is no reason to translate it.

?       This message will not be displayed if there is no
        text.
```

The translated **adv_txt** files must be located in the **~/config** directory and be named **adv_txt.***language*. We have included several language files in this directory. **adv_txt** is a fallback file, which is used in case no specific language file is available or if the message number is marked with an asterisk. When the user has not specified any language, the file **adv_txt.default** is used. This should be a symbolic link to the language normally used on your system. If the user has specified a different language with the **set/language** command, then the file **adv_txt.***language* is used.

Please check the files for completeness and send any corrections of it back to us to be included with future releases of CLX! Again, note that the messages having an asterisk are internal messages only and are always taken directly from **adv_txt**.

When you are translating or adapting a message file, it is convenient to have both the original and the translated messages near each other so that you can make a comparison. For this, we have included a tool called `check_adv_txt` in the `~/tools` directory. Use this tools as follows:

```
$ cd ~/config
$ check_adv_txt adv_txt.italian
```

With the command line switch `-m` you can force it to show only messages which are missing in the translated file.

3.6 Location data in ~/config/wpxloc.raw

The file `~/config/wpxloc.raw` originates from the PacketCluster software and contains a huge number of detailed location information for different DXCC countries, islands states and regions of the earth.

The synax of this file is as follows:

```
---------+-----------------------------------
  Column | Meaning
---------+-----------------------------------
       1 | Prefix or Prefixes
       2 | Full Country Name
       3 | DXCC country number
       4 | ITU zone
       5 | CQ zone
       7 | Time zone (negative means east)
    7-10 | Latitude (South is negative)
   11-14 | Longitude (East is negative)
      15 | Follow up flag (* - optional)
      16 | DXCC country flag (@ - optional)
---------+-----------------------------------
```

Comment lines are starting with `!`.

So the file looks like this:

```
!Updated many times by DJ3IW, DK2OY, DB0SPC 1991,1992,1993,1994,1995
!
1A S.M.O.M.-1A          268 28 15 -1.0   41 54 0 N 12 24 0 E     @
1B NoDXCC-(illegal)     288 39 20 -2.0   35 0 0 N 33 0 0 E
1S Spratly-Is-1S        269 50 26 -8.0   8 48 0 N 111 54 0 E     @
3A Monaco-3A            270 27 14 -1.0   43 42 0 N 7 23 0 E      @
```

As many grown-up PacketCluster Sysops have put considerable work into maintaining this file, CLX continues to use this data in the `SHOW/SUN` and `SHOW/HEADING` commands. However, the data is converted into a slightly different form and then called `~/config/location.dat` at the startup of CLX. `~/config/location.dat` has the following format:

```
---------+-----------------------------------
  Column | Meaning
---------+-----------------------------------
       1 | Prefix
       2 | Full Country Name
```

```
      3-5 | Latitude
      6-8 | Longitude
        9 | Time zone
---------+----------------------------------
```

Whenever `~/config/wpxloc.raw` is changed, however, `~/config/location.dat` will be created anew. There is nothing you need to do about this, excepct adding new countries and regions from time to time if you so desire. Do not touch `~/config/location.dat` as this will is generated automatically.

3.7 Special Characters in System messages and Help Files

CLX understands national characters and is able to transmit them depending on your character set. You must code the special characters according to the ISO 8559 conventions, that is `&auml;` for the letter ä etc. This should be familiar to everyone writing HTML code.

The special characters CLX understands are defined in the file `~/config/char_set`. In that file the corresponding character codes for the different character sets available must also be specifed. Currently we have four, the Latin-ISO (used by Linux and MS-Windows), IBM (used by MS-DOS), DIN (which is using the rare characters {, }, |, [,],\ and ~ for national character encoding — Baycom used this convention) and one with uses ASCII transcription.

Currently, the system messages in `~/config/adv_txt.default` (which is a link to your default language file), the output of external programs and the help files will be transposed.

To switch languages, users may use the `SET/LANGUAGE` command, to switch character sets use `SET/CHARSET`. For more help take a look at the user help files.

3.8 Message of the day or "notice"

CLX allows setting of a so-called "Message of the Day". This message is displayed to every user logging into the system. This feature makes it possible to announce meetings or special events to the users in a convenient way.

To enter a new message of the day, you should use the following commands:

```
set/motd

    Now enter the new message

/exit
```

A new file `motd` gets created in the `~/box/info/etc` directory.

It is possible to automatically set the MOTD with a program which is being called through `cron` every day at 00:01 UTC. An example of such a program is in the `mk_motd` script in the `~/tools` directory. This script calculates the date of the regular DX meetings of the B.C.C. in both Nürnberg and München and sets the appropriate message. This way, club members are reminded well before the meeting happens. Feel free to expand this script to make it useful for you. Some time in the future, I will make a more general tool - currently all is pretty hard-coded in that file.

New users, who have not logged in any time before optionally are displayed the file `motd_new` in the directory `~/box/info/etc` if it exists. A new user is a user without a record in the database.

3.9 CLX unavailable

To inform your users of the node being temporarily unavailable, there is a file which is displayed when `net_usr` does not find CLX running. First, `net_usr` tries to find `$HOME/config/clx_etxt`, then `/usr/local/etc/clx_etxt`. If both fails, it will display the message `clx unreachable`.

Also users coming in via the clx daemon `clxd` (TCP/IP) will be sent this message when CLX is down.

3.10 Multiple Connects

A word about multiple connects: CLX accepts multiple connects from the same callsign using different SSID's. In a CLX Cluster you may connect each of the single nodes using the same callsign (even the same SSID). However, connecting the same node twice with the same callsign is not possible. CLX is using the callsign as a communication channel identifier. What will happen, whenn you're connected as, let's say DL6RAI, and start up another session as DL6RAI? The old DL6RAI will receive the greeting message, the new DL6RAI will receive nothing. All commands he enters are answered to his first connection. And when he tries to log out, the nr. 1 connection is terminated.

3.11 Amateur Band Boundaries

Amateur band boundaries are defined in the `ar_band` database table. This information may change or may be different in your country. Make the appropriate corrections to the file `~/config/ar_band.cd` and then issue the following command (as `clx_us`, with `postmaster` running):

```
$ db_maint a
```

You may also add new modes and frequency boundaries. You must then follows these rules:

- Every distinct band must have a single `b_symb` attribute. The `SHOW/DXSTATISTICS` command uses `b_symb`.
- It can have multiple `b_syn` attributes. For example the 15-m-band can be addressed as `15` or as `21` in a `SHOW/DX` command. `SHOW/DX` uses `b_syn`

3.12 Configuring the AX.25 software

With the AX.25 utils after version 2.0.12, some changes were necessary from previous versions. Where CLX used to communicate directly with the kernel drivers, we have now `ax25d` as a super daemon launching applications such as `axspawn`, a personal messages system or — in our case — `net_usr` to connect to CLX. Please follow the instructions coming with the AX.25 utils package.

3.12.1 Using a KISS serial port

For a simple and plain CLX installation the following steps should be necessary:

1. edit `/etc/ax25/axports` and put your callsign for outgoing connects there. Give the interface a name which you will refer to later. I called mine "`kiss`".
2. edit `/etc/ax25/ax25d.conf` and put the following lines in it:

```
[kiss]
parameters   1 10   *   * *  0
default 7 10 2 900 * 15 - clx_us /usr/local/clx/bin/net_usr net_usr -x %s
```

This boils down to: Whenever someone calls DL6RAI-7 on interface `[kiss]`, start the program `/usr/local/clx/bin/net_usr` with the switch "`-x`" (which converts CR/LF) and his callsign including SSID as a parameter. The process is started under the UID of `clx_us`.

3. Make a startup script in `/etc/ax25` which is called when your system boots. My `ax25startup` looks as follows:

```
#! /bin/sh
set -x
echo -e "\015\015\033@K" > /dev/tnc
/usr/sbin/kissattach /dev/tnc kiss
/usr/sbin/ax25d -l
sleep 3
/usr/sbin/kissparms -p kiss -t 150
```

The `echo` command switches my TNC into KISS mode. Then `kissattach` is being started on the TNC device. Then `ax25d` is started with the "`-l`" option which makes the AX.25 daemon send log messages to syslog. Then we must wait a little before changing KISS parameters with `kissparms` as was necessary here (TX delay).

Also I have a script for shutting down the AX.25 programs:

```
#! /bin/sh
set -x
kill `/bin/ps aux | awk '/n\/ax25d/ {print $2}'`
kill `/bin/ps aux | awk '/n\/kissattach/ {print $2}'`
echo -e "\300\377\377\377\300" > /dev/tnc
```

3.12.2 Using AX.25 over Ethernet

Another way to use the AX.25 software is with the BPQ encapsulation. This allows us to send AX.25 packets via Ethernet and is compatible to Flexnet's `ETHER.EXE` module from DK7WJ (which additionally requires a packet driver) and probably software from G8BPQ. For this to work on the Linux side, you just add another line in the `/etc/ax25/axports` file looking like this:

```
[ether]
ether           DB0CLX-1        19200   255     4       ethernet
```

Additionally, you must configure the ethernet port, in most cases `eth0`:

```
# /usr/sbin/axparms -dev eth0 ether
```

And in the `/etc/ax25/ax25d.conf` file we should add the port `ether` to the configuration:

```
[ether]
NOCALL   * * * * * *  L
default  * * * * * *  - clx_us  /usr/local/clx/bin/net_usr net_usr -x %s
```

Now we can connect the CLX port DB0CLX-1 via ethernet.

3.13 WAMPES and CLX

3.13.1 Installing real WAMPES

At this point in previous releases of the CLX software, we had a chapter on how to get, compile and install WAMPES. We felt that this information was slightly outdated now. So we left it out. If you really have to use WAMPES for one reason or another, you will probably know better how to install it.

To make CLX run with WAMPES, you only specify one key word in the `clx_par` file, that is:

```
wampes:
```

You may, if desirable, specify a port number after the colon. This will allow you to change the port number which CLX uses for communicating with WAMPES. The default WAMPES port is 4718 and need not to be specified.

It is also possible to specify a WAMPES host if your WAMPES installation is running elsewhere. For this use the keyword

```
w_host:
```

in the `clx_par` file.

Due to WAMPES' philosophy, SSIDs are not reported down (or up?) to the application running on it. Due to this, when using WAMPES, node callsigns must be unique, there is no way to have DJ0ZY be a normal user and DJ0ZY-7 a CLX node. This is true only for incoming connects. When going out through WAMPES, you can chose any SSID you like.

This also leads to the problem that users' SSIDs are generally ignored and you cannot be connected to two adjacent nodes which have a link between them using the same root callsign.

This is a WAMPES "feature" and cannot be changed by CLX.

3.13.2 Using TNT's WAMPES socket feature

TNT, written by DL4YBG, is a very convenient packet hostmode program for Linux. New versions are available from *http://www.snafu.de/~wahlm* TNT runs on the Linux console as well as in an xterm under X11. One of the features of TNT is its ability to provide a WAMPES compatible socket interface for outgoing connects. This feature is also known as TNT's `NETCMD` interface. Along with the `autostart` feature, which makes TNT automatically start an application when a specific SSID is being connected from outside, TNT can be used as both an interface and a control terminal for CLX.

To enable CLX operation via TNT, you must specify the following two lines in the `tnt.up` file which is executed when TNT starts up:

```
socket netcmd *:4718 dl6rai-7
autostrt on
```

The first line makes TNT listen on port 4718. If a connection is made, the application may use the command "`connect ax25 db0aab`" for example to establish a connection to DB0AAB. You can check this out yourself by simply doing a "`telnet localhost 4718`".

The second line enables the autostart feature for incoming connects. If — for example — the channel with the callsign DL6RAI-7 is being connected by DK2OY, TNT will set the environment variable `$CALLSSID` to `dk2oy` and spawn a shell running the script `clx_sh`. For this you create a file called `autostrt.tnt` in the TNT directory and add the line:

```
dl6rai-7 run clx_sh
```

to it.

The file `clx_sh` is a shell script located in `.../tnt/bin` (or whichever is defined as the TNT `run_dir`) which should contain the following two lines:

```
#!/bin/bash
exec /usr/local/clx/bin/net_usr ${CALLSSID}
```

Also you must have some channels programmed with that very callsign you like to use. You can do that by putting several lines like

```
channel 1
mycall dl6rai-6
channel 2
mycall dl6rai-6
...
```

into your `tnt.up` file.

All of this is a little confusing. Again, these are the steps you have to do:

1. Make TNT listen on a socket port
2. enable the autostart feature
3. Create `autostrt.tnt`
4. Create a script called `clx_sh` in TNT `run_dir` (default: `.../tnt/bin`)
5. Have some packet TNT channels programmed with the callsign you want to be connectible under

When using the `w_host` variable in `clx_par` and setting the `$CLX_HOST` shell variable or calling `net_usr` with the "`-h`" switch on the remote host, you might even have CLX and TNT running on different machines. DG4MMI reports that he got TNT and CLX running this way very nicely. DL1RF reported good success with TNT 1.1a3, DPBOX and CLX 2.07 all running on the same machine. `net_usr` is a statically linked program, it only needs `libc.so.5`. So the program may be used on a different computer.

The advantage of TNT+CLX is that you get control over each communication channel as you can see what's going on by switching TNT channels. You may watch the PC-PC communication going on or you can see how and why routing scripts work or fail. Generally it gives you more control over your CLX installation and makes it more transparent.

Thanks to DL4YBG for making this feature available!

DL4YBG has slightly extended the WAMPES socket interface so that after the target callsign, one can specify the source callsign to be used on an outgoing connect through TNT.

The new syntax is as follows:

```
CONNECT <transport mode> <destination callsign> [source callsign]
```

If no source callsign is specified, the default callsign is taken from the `NETCMD` interface.

Additionally, when starting an outgoing connect via this interface, TNT uses its own routing table and after establishing the connection activates the socket interface. As a side note, this also automatically changes the

source callsign of a AX.25-connection with this callsign already exists to the destination. So you just need not care about it at all.

The setup of the routing file (`routes.tnt`) is a little tricky. You must make sure that the destination callsign again shows up with a `T>` tag at the end, like in the following example:

```
T>DB0SUE-7 CLX; F>db0aab N>db0bro N>db0il T>db0sue-7
```

If it doesn't, then CLX will never receive the final connect message and the connection will not be established. When setting this up first, try making a connect manually using `:XCONNECT` and watch the upper status line switching to the destination callsign when that is reached. In the above case it should look like this:

```
Ch:05 Stat: IXF wid DB0SUE-7
```

3.14 Making CLX access available through telnet

With version 2.03 a new program `clxd` was included with the CLX software. This perl script allows you to connect directly to CLX via telnet (without first having to log in). This is achieved by putting the following entry into `/etc/inetd.conf`:

```
clx stream tcp nowait root /usr/sbin/tcpd /usr/local/clx/bin/clxd clx
```

Also you must add to your `/etc/services` file something like this:

```
clx             41112/tcp               # CLX Login
```

To make it really work, you must create an emtpy file called `passwd` in the directory `~/config`:

```
$ touch ~clx_us/config/passwd
```

This is all what you have to do. Now you must kick `inetd` to make it read its configuration file:

```
# ps aux | grep inet
root       109  0.0  0.4   832    68  ?  S   12:27   0:00 (inetd)
# kill -1 109
```

Users can now log in via port 41112 with the command:

```
$ telnet <your hostname> 41112
```

This is what will happen:

```
Trying 192.0.0.2...
Connected to al7nj.
Escape character is '^]'.
Welcome at the clx gateway - you logged in from 192.0.0.1/1028.
```

Now you must enter your callsign. If you log in for the first time (i.e. `~/config/passwd` contains nothing), a new password will be asked for.

```
Enter new password:
Reenter password:
```

So a new entry is created in the `passwd` file. Otherwise you must enter the password which was used previously.

```
Password:
```

The echo will not be suppressed so be careful when someone is watching you. I did not figure out how to do this in a clean way. However, the program tries to make sure that a valid amateur callsign is being used for login:

```
Your callsign: wrdfl
Callsign invalid.
Your callsign:
```

The `passwd` file contains four fields, separated by colons. It contains callsign, password (encrypted), the IP address from where the connect was made and date/time of the first login.

```
dl6rai:wroX0dhQ.42t:192.0.0.2/1044:27-Feb-96 2023z
df2rg:wr9y/v96xsF9:192.0.0.1/1029:29-Feb-96 1640z
```

This is a very rudimentary security check and we do not recommended that you put this on the Internet right away.

After successfully logging in, you will be treated like any other normal user. You can talk, you receive DX spots etc. There is no difference between normal users and users coming in via `telnet`.

In fact, you can also use this feature to provide an incoming node connection. You would need some interface to get over the login mechanism however. This could be done by a connect script described in section 3.16 (Connect Scripts).

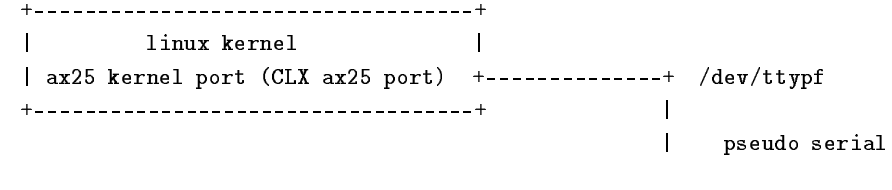
As a special rule, clxd turns off the telnet echo when a known node callsign connects. This avoids problems due to reflected PC10 frames (looping talk messages) and circulating mail.

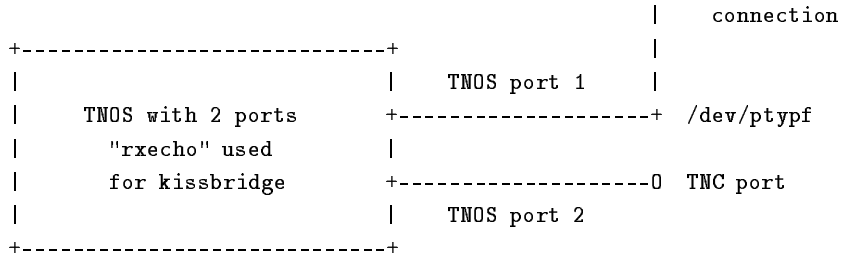
3.15 Using CLX with TNOS

CLX can be used with TNOS. The following information was received from Joni Bäcklund, OH2NJR and Andrea Fiorentino, N5KME/I0:

OH2NJR: "I just don't see any reason to use TNOS with ax25 kernelcode anymore. With the latest ax25 kernel (2.0.30 with module13 and latest ax25 utils) I can do everything that TNOS does and save about 1.5MB of memory at the same time."

The thing was done by attaching an ax25 kernelport to a pseudo ax25port on TNOS and then connecting the real TNC into another port in TNOS. Then we only had to use the `rxecho` command with TNOS to copy pakets from Linux's TNOS port to the TNC-TNOS port and vice versa. This was the way I used TNOS and Linux/CLX."





It is done by using pseudo TTYs as "wired serialports". The kernel uses `/dev/ttypf`, TNOS uses `/dev/ptypf` to connect the kernel and after that TNOS uses another port for the KISS TNC etc. Take a look at OH2RBJ's current setup at *http://www.clinet.fi/~njr/rbj.html*

Here is some information frm I0/N5KME:

1. attach a pseudo kiss device with `kissattach`:

   ```
   kissattach /dev/ptypf link
   ```

   `link` is defined in `/etc/ax25/axports` as:

   ```
   # <port name> <call> <speed> <paclen> <window size> <description>
   link     N5KME-6 38400    255          7        clx <-> tnos
   ```

   Note that the call must be the same as in CLX

2. configure clx for the correct device and speed in `~/clx_us/config/clx_par`:

   ```
   tnc_dev:      /dev/ptypf
   tnc_baud:     38400
   ```

3. configure in TNOS the link with CLX in the file `autoexec.nos` add:

   ```
   attach asy ptypf - ax25 clx 2048 1024 38400
   ```

4. Make the CLX port see everything a real radio port sees. My radio port for the PacketCluster link is named `uhf`.

   ```
   ifconfig clx rxecho uhf

   ifconfig uhf rxecho clx
   ```

   Note: **Only one** radio port can be `rxecho`ed or the CLX link will see a tremendous confusion.

3.16 Connect Scripts

To make establishing node connections under CLX very flexible, we included a connect script feature as a means to improve outgoing links to other nodes and sources of DX information. The concept is built around a program (written in any language) which establishes a connection to a remote node and then, when the connection is established, switches into transparent mode. CLX uses pseudo TTYs as an interface to the scripts.

Note that in some Linux distributions today, the `/dev/pty*` and `/dev/tty*` device files belong to user `root` and group `tty`. To allow CLX accessing the pseudo-ttys you must either make `clx_us` member of group `tty` (best solution) or change the ownership of the device files.

We have found the Expect package from Don Libes (available on the Internet on any Sunsite mirror in the directory `/devel/lang/tcl`) to be an ideal platform for such scripts as it already contains a great number of pattern matching functions and through its `interact` command is able to keep the connection alive after the script is done.

I downloaded the Expect package from *ftp://ftp.leo.org/pub/comp/os/linux/sunsite/devel/lang/tcl/expect-5.18bin.tgz*. Nowadays, Expect is usually available as part of the Linux distrubition and normally installs under `/usr/bin`.

Mike Sims, KA9KIM, who installed Slackware 3.5 in October 1998 reported that the version mentioned above is kind of old. Presently, the expect package is located at the `expect.nist.gov` web site. However, the newest version also needs Tcl version 7.4 which is also not included with that distribution so he had to get the Tcl 7.4 sources too and then recompile the `expect` package.

The conditions for the external programs building up the connection are simple and straightforward: After being called from CLX the program must either exit if the connection cannot be established, or the first line of text coming back through is understood as a successful connect.

Let's look at some examples which we have tested.

3.16.1 Shell Scripts as Connect Scripts

The first approach is a two-liner Bourne shell script. It uses the program `call` from the AX.25 utilities to establish the connection to F6KNL-3. This example was worked out together with Lucien Serrano, F1TE, to connect to F6KNL-3 via F6KNL-9 and a Rose node. The script is very simple, it does not even require Expect as there are just two commands:

```
#!/bin/sh
read $something
/usr/bin/call -r ax0 f6knl-3 f6knl-9 833501
```

This script is a plain shell script as you an see in the first line (the `#!/bin/sh` means to the shell: start the program `/bin/sh` and then feed it with the rest of this script). The `read` command is necessary because CLX will send the string "`connect ax25 f6knl-3`" to the script after its start and this should simply be dropped and not screw up the next program or the remote node.

However, in a more general script you could make use of this information. With CLX version 4.00, the script receives command line parameters so now it is possible to provide you with a connect script for all occasions:

```
# These parameters are taken from the command line
PROG=‘basename $0‘
MYCALL=$1; shift
PORT=$1 ; shift
DIGILIST="$*"

# Now we go process the 3rd parameter from STDIN
read conn
set -- $conn
logger -p local5.info "$PROG: /usr/sbin/ax25_call $PORT $MYCALL $3 $DIGILIST"
stty -echo raw
exec /usr/sbin/ax25_call $PORT $4 $3 $DIGILIST
```

The script is called with the outgoing callsign as the first parameter, followed by any additional parameters in the `conn_path:` entry for that destination. Note that `ax25_call` does not handle end of line characters and so it may not be working for you unless you modify the source code and compile your own ax25_call.

After the script has started, it receives a line from `con_ctl` specifying the destination call as the third and optionally the outgoing callsign as the fourth parameter. So if the file `~/config/cluster_par` contains these lines

```
SECTION:        db0bcc
conn_call:      dl6rai-7
```

then `con_ctl` will in fact send the string

```
connect ax25 db0bcc dl6rai-7
```

to the connect script.

These scripts work well in reality and are pretty straightforward and simple to understand. They have some disadvantages though:

- There is no internal timeout
- It will make CLX believe that the connection is established before it really is. CLX understands that if anything is received from the script, the connection has been established and any watchdog timers are started.

3.16.2 Using TCP/IP for a connect

A rather elegant usage of `net_usr` allows linking CLX nodes via TCP/IP. `net_usr` has the `-h` option to allow specifying a CLX host. So `net_usr -h al7nj dl6rai-7` will start a connection to CLX running on `al7nj`. So taking the same configuration as above, a simple script to connect another CLX node via TCP/IP would look like this:

```
#!/bin/sh
read conn
set -- $conn
net_usr -h $3 $4
```

This assumes that the callsign you are connecting to (`al7nj`) is also a known IP hostname (by having it in your `/etc/hosts` for example).

3.16.3 A simple Expect Script

Here is another example, a little more complicated as it involves several steps. As a first step it connects DB0AAB and then tells DB0AAB to setup a link to DB0BCC. Finally the program goes into transparent mode with the command "`interact`". If either the first L2 or the L3 connect fail, the program aborts ("`exit 1`" in the code). In the first line again you can see that an interpreter is called, this time the program `/usr/bin/expect` with the flag "`-f`".

```
#!/usr/bin/expect -f
#
# Routing through Kernel AX.25
#

# --------------------- User configurable ---------------------
set digi db0aab
```

```
set port kiss
set destination db0bcc
set l2_timeout 10
set l3_timeout 30
# ---------------------- User configurable ---------------------

set timeout 0
log_user 0

set timeout $l2_timeout

spawn /usr/bin/call -r $port $digi
expect "***Connected"

expect timeout {
        send_user "\n### timed out waiting for '=>'\n"
        exit 1
} "=>"
send "c $destination\n"

set timeout $l3_timeout
expect timeout {
        send_user "\n### timed out waiting for '*** connected to'\n"
        exit 1
} "*** connected to"

send_user "*** connected to $destination\n"
interact
```

Of course, the parameters in this file can be left out as you can specify them in the configuration file. Let's look at another more complicated script.

3.16.4 A more sophisticated Script

Here we dial up an Internet connection through `cu`, login, telnet across the Internet and login at the remote `clxd` port. We have not tested this script, so it may or may not run. But we hope you get the idea behind it.

```
#!/usr/bin/expect -f
#
# Routing through cu, telnet
#

set timeout 60
log_user 0

# Use cu to dial up my Internet Service Provider
spawn /usr/bin/cu isp
expect timeout {
        send_user "\n### timed out waiting for 'Connected.'\n"
        exit 1
} "Connected."

# Unix Login
```

```
send "\n\n"
expect timeout {
    exit 1
} "ogin:"
send "name\n"
expect timeout {
    exit 1
} "sword:"

# Wait for shell prompt
send "MyPaSsWoRd\n"
expect timeout {
    exit 1
} "\$"

# Now telnet to my partner's clxd port
send "telnet 193.0.23.245 41112\n"
expect timeout {
    exit 1
} "Your callsign: "

# Login at the remote clxd
send "xx0xx\n"
expect timeout {
    exit 1
} "Password:"
send "my_login_passwd\n"
expect timeout {
    exit 1
} "*** connected to clx"

# OK, we're done
send_user "*** connected to $destination\n"

# Transparent mode
interact
```

3.16.5 A virtual one-way connection

As a last example, think of a program which does nothing else but listen to an AX.25 channel where PacketCluster spots are being transmitted. On a busy channel, some DXers are always maintaining a connection to some PacketCluster system. Or you could have a listen on 14.096 MHz where some PacketCluster activity is going on. Just imagine such a program as an interface to CLX. From within CLX you would simply configure this as a "virtual" connection to another node.

The program `/usr/bin/listen` from the AX.25 utils package allows us to listen to the different ax25 ports. With the "`-r`" option it outputs the monitored frames in a readable form which can then be scanned for any DX spots. Note that the program listen can only be started with root rights, so either you will have to set the suid bit on the `/usr/bin/listen` binary or become root.

Here is the script:

```
#!/usr/bin/expect -f
#
# Listening on my radio port for spots
```

```
#

log_user 0
set timeout -1

spawn /usr/bin/listen -r

while (1) {
        expect -re "\[^\n]*DX de .*\[^\n\r]" {
                send_user "$expect_out(0,string)\n"
        }
}
```

This simply starts the program **/usr/bin/listen** and goes into filter mode. The complicated looking expression

```
\[^\n]*DX de .*\[^\n\r]
```

is nothing else but a definition for a DX spot in expect's regular expression form.

For more information about expect, I recommend reading the book "Exploring Expect" from Don Libes, O'Reilly & Associates, Inc., ISBN 1-56592-090-2 available at any good computer book store.

Think of other ideas like this:

- use a packet program and write all DX spots into a file. Then use **tail -f** to read from this file. We have included a script called **file_monitor** in the **~/exec/connect** directory.
- use TNT's **:logmon** command to log anything to a **pty** as follows:

  ```
  :logmon /dev/ptyqf
  ```

  Secondly use a script containing

  ```
  cat < /dev/ttyqf
  ```

  so it will read anything which comes in from the monitor.

3.17 CLX and Callbook Data

Several interesting address sources are available on CDROM as well as on-line on the internet. Buckmaster Publishing was first to provide access for the Pavillion software to their address database. CLX does support the **QRZ! Hamradio CDROM**, the International (Flying Horse) Callbook CDROM and also a free data format into which you can configure your data.

3.17.1 The QRZ! Hamradio CDROM and other callbook data

From version 2.00 CLX allows you to use any callbook data available on your system. This feature enables you to directly access the QRZ! Hamradio CDROM from Walnut Creek (AA7BQ) and also your own data files if they are supplied in one of two data formats.

A new directory **~/callb** must be created. This directory may contain any further subdirectories with callbook data. The data in this directory must comply with one of these formats:

1. Sorted by suffix:

```
AA0A  ,Name.....,,
AB0A  ,Name.....,,
AC0A  ,Name.....,,
AD0A  ,Name.....,,
...
AA1A  ,Name.....,,
AB1A  ,Name.....,,
...
AA0AA ,Name.....,,
AB0AA ,Name.....,,
AC0AA ,Name.....,,
...
K 0AA ,Name.....,,
K 1AA ,Name.....,,
K 2AA ,Name.....,,
...
```

2. Sorted alpha-numerically:

```
AA0A  ,Name.....,,
AA0B  ,Name.....,,
AA0C  ,Name.....,,
AA0D  ,Name.....,,
...
AA1A  ,Name.....,,
AA1B  ,Name.....,,
AA1C  ,Name.....,,
...
AA0AA ,Name.....,,
AA0AB ,Name.....,,
AA0AC ,Name.....,,
...
```

Missing characters are replaced by blanks. Fields are separated by commas. The sort type is indicated by file name.

```
<name>.s        sorted by suffix
<name>.c        sorted alpha-numerically
```

No other formats are available at this time. More to be done in the future.

Any available sources are supplied in the `clx_par` file. With the `callb` parameter you may specify any number of directories to search for callbook address information. The software searches each directory until the requested address is found. Then it stops. This way you may have directories with descending priority, for example a local directory where you put in very reliable information and as a second source the CDROM information. If the information is found in the local directory, the (probably wrong) information on the CDROM is not shown to the user.

There may be any number of files in the `~/callb/*` subdirectories. The information must be mutually exclusive, i.e. you cannot have two files with Italian addresses in the same directory. This must either be in a single file or in two different directories.

CLX creates two index files in each subdirectory, one for the suffix oriented files and another for the alpha-numerically sorted files. These index files are being created when starting CLX. The indexing runs in the background so it might take a few minutes before you can access the data after the first start.

Whenever information is changed in these directories, CLX will realize that something has changed and rebuild the index files. The index files are named `clx_idx.s` and `clx_idx.c`. Their size is approx. 200 kB each.

The files may not be in that directory physically. It is OK to make symbolic links to the physical files, thus allowing you to keep your CDROM mounted under `/cdrom` or whatever.

Example:

You have two directories under callb:

```
local
qrz_dx
```

`local` contains any local addresses, probably gained and converted from a club's roster, your private address database or something. So there is only a single file in local, named `address.c`. This file must be sorted alpha-numerically.

The `qrz_dx` directory contains a link to the QRZ! Hamradio CDROM mounted under `/cdrom`:

```
all.s -> /cdrom/callbk/callbkc.dat
```

You can even keep the files on a different machine and mount them via NFS. We have put a sample file under `~/callb/local` which is named `bcc.c`. It contains some entries from the BCC's club roster. This file will be used by CLX if you specify the directory

```
callb:  local
```

in the `clx_par` file.

The user command to access the information is `show/callbook`. This is an internal command and it cannot be changed. The `show/address` command which was used here before is now free for your own database, probably importing from a PacketCluster `.ful` file (see also 5.9.3 (read-ak1a)).

3.17.2 The Flying Horse CDROM Callbook

The data on the Flying Horse CDROM Callbook is provided through the use of a DOS program named **calldos.exe**. This program and a little Expect script is used to gather the data from the CDROM. To get this working, you must first get the DOS emulator going, preferrably by reading the supplied documentation and the DOSEMU-HOWTO document, available, for instance, from *http://sunsite.unc.edu/mdw/HOWTO/DOSEMU-HOWTO.html*.

You must be able to start the DOS Emulator and access the Callbook data from a drive letter. This drive letter must be defined in the CLX configuration file `clx_par`. Additionally, no printer must be defined in DOSEMU, as the program supports only `LPT1:` and we need to send its data to a file.

If all is working well, you could try running the program first from the `clx_us` shell prompt:

```
$ ~/exec/command/show/cba dl6rai dk2oy
```

and you should get:

```
Manfred Petersen, DK2OY
Hardtstr. 83
D-40629 Duesseldorf
Germany
```

The user command to access the Flying Horse callbook data is `SHOW/CBA`.

3.17.3 Using online address data from the Internet

Thanks for Matthew George and Erik Olson there another script is provided in the `~/exec/command/show` directory allowing online access to Internet address data directly from `www.qrz.com`. You must have a continous Internet connection. The script is provided "as is", I have not tried it here but it is used in a production mode at NC7J. The script is named `cba.internet` and to make it easy for your users, you should rename `cba` to `cba.cdrom` and `cba.internet` to `cba`.

4 CLX Programs, directories and database tables

4.1 Files and Directories of CLX

CLX uses a specific file structure under its home directory `~clx_us`:

```
ax25/                   some binaries and AX.25 .tgz files
backup/                 where tools/bup_db writes and reads ASCII backups
bin/                    CLX binaries and clx startup/shutdown script
box/batch/start/        user startup files (profiles)
box/iclb/               mail to be forwarded
box/mail/               user mail directory
box/bulletin/           DX bulletin directory
callb/                  callbook files, links to QRZ-DX CDROM
exec/command/           command extensions for all users
exec/connect/           connect scripts for CLX
exec/interpr/           command extensions for all, with message interpreter
exec/privileg/          command extensions for admin users
exec/checks/            directory for mail checking
config/                 configuration files
db/                     command files to create database tables
box/info/help ->        Link to default language help files
box/info/help.german/   German user help files
box/info/help.english/  English user help files
box/info/help.french/   French user help files (tnx HB9BZA)
box/info/help.italian/  Italian user help files (tnx IK3HUK)
box/info/help.portuguese/       Portuguese user help files (tnx PP5AQ)
log/                    directory for log files (specify in /etc/syslog.conf)
tools/                  system and user administration tools
```

4.2 Programs

The CLX software consists of several programs which are linked via shared memories and database entries. The server software is being started and shutdown by the `clx` script, which starts and stops the processes.

4.2.1 Server programs

Here is a definition for each particular module

con_ctl

is the communication interface (AX.25, telnet sockets)

rcv_ctl

manages receive spooling.

snd_ctl

manages transmit spooling. At a later time, this process may handle priorities. Currently all messages are in one single processing queue.

rm_disp

receive message dispatcher, forwards messages to other processes.

usr_req

manages all user commands, user database requests, forks a process for every database task to make it run in the background. The background process later sends the results directly to send_ctl.

mb_ctl

mailbox control. The mailbox keeps messages in a file system, header information is stored in the database.

usr_ctl

user administration (logins, logouts, logbook).

usc_mng

manages external user commands and command extensions.

usr_dlg

manages user dialogs like the `set/priv` sequence.

bbs_if

Packet Radio Bulletin Board System (PBBS) interface implementing the W0RLI mail forwarding protocol.

iu_com

inter user communication (talk, conference).

icl_com

inter node communication (processing and generating PCxx messages).

clx_ctl

system administration, installing shared memory pages. It also controls cyclic jobs like building up link connections, testing with ping etc.

int_com

manages internal program communications.

udt_mng

manages private data tables (udt's).

db_*

special program started for special access to database. This program will be terminated when the transaction is finished.

clb_sad

Manages the index files in the callbook directory `~/callb`. This program will be started at the start of CLX and then finishes, after the update of these files. When a user accesses the callbook, this program handles the interaction between user and callbook data.

bbs_if

a program which is started up for each incoming PBBS connection. PBBS systems are identified using the `bbs_lst` configuration command in `clx_par`. See 3.3 (bbs_lst) for further details.

4.2.2 User Programs

The following programs may be run from the console:

net_usr

A login program for clx from the console or when started via ax25d. `net_usr` is a statically linked program (except that it needs libc.so.5). It can be run from another computer and does not need any of the specific CLX shared libraries.

`net_usr` has a number of command line switches, enabling different features:

```
-h <host> connect to CLX on a different host

-i      necessary for batch programming. Makes net_usr read from STDIN
        only when the connection to the running CLX is established.
        This should make it possible to run net_usr from a shell
        script reading commands from a file (like this):.

            net_usr -i dl6rai <<NNNN
            sh/dx/10
            bye
            NNNN

        Unfortunately, due to CLX's multiple processes, the second
        command goes through faster and so the output of the first
        command never gets out. We have not found a solution to this
        problem at the moment but it did not seem very important
        anyway. The general problem is that CLX does not serialize
        the commands but tries to work them down as soon as possible.

-r      similar to the -i option above with the difference that the
        program only terminates after having received a response from
        CLX. It is working for an announce command like this:

            echo "ann DB0BCC will be shut down now" | net_usr -r

-x      Makes net_usr convert all <CR> characters to <LF> on input.
        This is necessary for net_usr to run correctly under ax25d.

-m      MS mode: Makes net_usr convert all <LF> to <CR><LF> on output.
        This is necessary to make telnet access under MS-Windows look
        nice.
```

term_usr

Another login program with readline support, basically providing the same functionality as **net_usr** but with GNU readline support. **term_usr** supports exactly the same command line options as net_usr.

Press "ctrl-P"/"ctrl-N" for previous/next commands, goto begin or end of line with "ctrl-A"/"ctrl-E" and move the cursor back and forth with "ctrl-B"/"ctrl-F". The standard cursor control keys may also be used.

Vi fans may also permanently switch to Vi mode by creating a file **~/.inputrc** containing the line:

```
set editing-mode vi
```

Then command history and recall are available with the standard Vi functions like **<ESC>-k**, **<ESC>-j**, etc.

term_usr supports a number of so-called inline commands for uploading or downloading of files. This way it is easy to start writing to a log file to document a CLX bug or you may use it for uploading files and data to the CLX system. Commands currently supported are:

```
~p              print working directory
~l              list current directory
~c <path>       change working directory to <path>
~> <file>       upload <file>
~< <file>       download to <file> (appends, never overwrites)
~.              finish download
~?              shows a short command description
```

When logging to a file, all input lines are logged with leading >>'s.

clx_adm

Administration tool which is being described in section 5.14 (Administration Tools) further below.

4.3 Database tables

CLX runs uses a number of tables in the Postgres database. The tables are accessed through the CLX programs so the user normally doesn't even know that there are databases, tables, indexes etc. For the Sysop however, it may be interesting to know that there are in fact a number of tables which are read from and written to by several CLX programs.

Table	Content	write access	read access
ann_data	Announce	icl_com	db_san
ar_band	Band Boundaries	<setup>	usr_req
distr_li	Distribution Lists	?	?
dx_data	DX-Data	icl_com	db_sdx
dxcc_pfx	DXCC Prefix List	clx_ctl	usr_req
dxcc_dat	DXCC Countries List	clx_ctl	usr_req
ml_dir	Mailbox Directory	mb_ctl	mb_ctl
ml_file	Mailbox Files	mb_ctl	mb_ctl
qsl_mng	QSL Information	usr_req	db_sqsl
sys_dat	System Information	clx_ctl	clx_ctl
us_data	User Data	usr_ctl	usr_ctl
us_log	Logbook	usr_ctl	db_slog

```
us_uhn   Homenode Table         icl_com,usr_ctl usr_req
www_data WWV Propagation Data   icl_com         db_swwv
wcy_data WCY Propagation Data   icl_com         db_swcy
-------------------------------------------------------
```

The QSL database is currently empty. **read_ak1a**, a tool to import QSL information from PacketCluster files is in the **~/tools** directory. Look at section 5.9.3 (read_ak1a) to find out more about this tool.

You can import a full-blown PacketCluster QSL-Database, like the one available from DB0SDX/DL1SBF which is available on the Internet, currently at *ftp://ftp.grossmann.com/pub/db*. This database is maintained by DL1SBF, who has been working on the data for several years - probably a good start for you.

The VHF Group DL-West has produced a fine database of VHF station information for central European VHF/UHF enthusiasts. Several interesting information from active callsigns are registered there like QTH locator, Packet and E-Mail address, Name, type of activity etc. This database can also be imported using the **read_ak1a** utility. See 5.9.3 (read_ak1a) for an exact description of the details. The command to read in the database goes like this:

```
$ read_ak1a -c -t vhf vhf_150.dbf
```

The database is available directly from Guido, please mail him at *dl8ebw@qsl.net*.

DL3KDV has taken the IOTA list and made it available for use with CLX. The **read_ak1a** utility can also import this type of data, automatically creating a table which shows entries by IOTA designators (like EU-001), Prefixes (like G) and continents (like EU).

The DXCC table is read from the file **~/config/cty.dat**. This file is a standard list of DXCC and WAE countries very popular and kept up to date world wide by CT (copyright K1EA) users. A recent version is always available from *http://www.contesting.com/ct/files/#cty*. Whenever this file changes, **clx_ctl**, which is the first program started when launching CLX, will reread the table in the background.

The home node table of CLX remembers user's home nodes and sends mail messages that way. The user may either explicitly select a home node location. If he does, the information is saved permanently, regardless of logins at other places. If no home node information is entered by the user, the system takes the first login it gets to know about this user, as his home node. If the user later connects to another node, CLX waits until it has received this information five times consecutively, before it does to change the home node information accordingly.

Some of the tables have indexes:

```
Table           Index file(s)
-------------------------------------------------------
ann_data        idx_and
distr_li        idx_dli
dx_data         idx_dxc, idx_dxl, idx_dxf, idx_dxd
ml_dir          idx_mdd, idx_mdn
ml_file         idx_mfn, idx_mfd
qsl_mng         idx_qsl, idx_mng
us_data         idx_usd, idx_uld
us_log          idx_usl, idx_ulc
us_uhn          idx_uhn
www_data        idx_wwd
wcy_data        idx_wcd
-------------------------------------------------------
```

All files of the CLX database are stored in the directory **$PGDATA/base/clx_db**.

It may be of interest to you that the hardest part for the database is retrieving "the last n" spots from the `dx_data` table. The reason behind this is that a query is built step by step. So when you query the last 5 records from the `dx_data` table, the following happens:

- Postgres would select all records, convert them to ASCII and write them into an intermediate file.
- then CLX would filter out the last five

The same is true for the last 5 records of 20 meters:

- Postgres would select all 20 m records, convert them to ASCII and write them into an intermediate file
- then CLX would filter out the last five

The more specific the query is, the less overhead. So when you look up the last KL7 station on 40 meters RTTY this job will be handled very quickly and efficiently. This is a general problem and has nothing to do with Postgres but with the use of a database in this way.

To get around this problem, we have taken two approaches:

- The last 100 spots are always kept in memory. This is where your "`sh/dx/5`" command is taken from and this explains why it is so much faster than "`sh/dx 2`". fed from.
- When querying the `dx_data` table, we first try to find a result in a small time interval going back from "now". If this fails the interval is made longer and longer until the query really looks through all data. Currently we have implemented 6 different intervals:

```
  2 days back
  8 days back
 31 days back
125 days back
500 days back
all
```

When the query gets into the third level (31 days) it writes a message to the user saying "`Patience. This query needs time`" (message #205 in `adv_txt`).

4.3.1 User defined database tables (UDTs)

With version 2.04 and later of CLX, it is now possible to install your own database tables in the Postgres database. For this, a conversion program called `read_ak1a` is provided in the `~/tools` directory and a set of CLX commands for the installation and administration of these tables exists (see 5.10.3 (udtadmin)).

4.3.2 Administration commands

This is the command set for administering the UDTs:

4.3.3 Remote database access

As a very esoteric feature, a database may be defined as follows:

```
create/!udt <tablename>/<flags> <clx command>
```

The "!"-character denotes that the record is not being looked up but a CLX command is called when the database is being accessed. This way, CLX can make its database tables accessible to other nodes. They can access the database table through a PC44 (remote database request).

Also, this provides a way to make an alias for a database table.

Let's say, you with to export your QSL database to another node (be it CLX or Pavillion PacketCluster).

```
create/!udt clx_qsl/+- sh/qsl <?>
```

This command creates a new table called `clx_qsl` which turns any request into the clx command `sh/qsl`. Note that the argument to the command is passed in the <?>. This must be specified, otherwise `sh/qsl` would be called without an argument.

Not only can you make an executable table but also records in your table may contain executable statements. This allows you to collect different programs and utilities under a common command interface. The syntax here is similar:

```
update/<tablename> !<key> <clx command> <?>
```

The <?> argument is optional and if stated will contain the command line parameter.

4.4 Files under the ~/box directory

The `~/box` directory contains a number of subdirectories where the part of the user data is kept, which is not stored in the Postgres database. These are mail messages, mails in the forwardig queue, user startup files, DX bulletins and the user help files. This portion of the file system is administered internally and you should use the Unix commands only in special cases where you know what you are doing. Generally, each file in this hierarchy also has a corresponding database entry and it's always a bad idea to get the two out of sync by manually fiddling around here. CLX provides a command interface to this area with the privileged CLX commands `PUT`, `GET`, `LS` and `RM` commands.

The consistency of the `~/box` directory hierachy is checked with the script `mbx_chk` (see 5.9.3 (mbx_chk)), normally at CLX startup. However, can also be called from the commandline.

4.4.1 The batch subdirectory

This directory contains another subdirectory called `start`. Below this, files are stored named after users' callsigns containing the startup procedure they have defined using the CLX command `UPDATE/PROFILE`.

4.4.2 The bulletin subdirectory

This directory contains a subdirectory for every year. These are generated automatically when an upload ocurs in a new year or if a user explicitly specifies the year with the `UPLOAD` command. The files are then stored one level below. There is no strict convention on how users may name the uploaded bulletin files.

The bulletin subdirectory is never changed by `mbx_chk` - but `mbx_chk` picks up any new files in this directory and includes them in the database. Knowing this, you may add bulletin files at the shell prompt to these directories by directly copying them into place and then running `mbx_chk`.

4.4.3 The iclb subdirectory

The `iclb` directory contains subdirectories for every partner node it forwards mail to. Under these directories you will find files named by a number ranging from 1 to 9999. These are files to be forwarded and the to be deleted after sucessful completion. The destination address and other message-relevant information is kept in the database.

Files in this directory are checked both ways by `mbx_chk`. Whenever the file or the database entry is missing, the corresponding part will be deleted. With the command `rm_fwd` you may remove mail forwarding queues directed to specific nodes. For more information on this command, see section 5.9.3 (rm_fwd).

4.4.4 The info subdirectory

The info subdirectory contains both a place for the help files (different languages) and for changible system files (like the motd file) in `~/box/info/etc`.

`mbx_chk` does not check this directory at all.

User help files are to be found under `~/box/info`. The old place `~/doc/user` is now used for the CLX user manual which was compiled by Ian, G0VGS, with a lot of effort. There is a printable version and also a HTML version of the user manual available.

The file structure in that directory is as follows:

The directory `help` should always be a symbolic link to the English version of the on line help files. These are the most current files.

Secondly, a link should be made from the language used in your country to `help.default`. This link is automatically created when you have defined a `language` parameter in the `clx_par` file. You should see that the help files in that directory are well up to date with respect to the English versions too. Additionally, there are a number of language directories like French, Italian, German and Portuguese. Leave them in place, if a user choses to switch languages, these directories will be used.

If you put some work into the help files, please be sure to mail us a copy to include in a future distribution of CLX. If you are translating the help files, please be sure to take a look at section 3.7 (special characters).

4.4.5 The `mail` subdirectory

The `mail` directory contains subdirectories for every user and for standard bulletin addresses. These subdirectories then contain files numbered from 1 to 9999 which are the true message files without and additional information (no headers). That information is only kept in the database.

Files in this directory are checked both ways by `mbx_chk`. Whenever the file or the database entry is missing, the corresponding part will be deleted.

5 System Administration Tasks

5.1 Time

CLX needs time in UTC. If you run CLX on the air, you should get this correct. Otherwise, your node will send out DX spots with wrong times. Look for information regarding local time settings in a file sometimes called `/usr/lib/zoneinfo/time.doc`.

Generally, your CMOS clock should run on UTC, also your system clock should be on UTC. If your machine runs completely on UTC, you need not change anything. However, if you like to have clocks display local time on your computer, you must read on.

First, in the `/usr/lib/zoneinfo` you should make a link from your local timezone to `localtime`. Additionally, you must make a link to `posixrules`, as this is needed to interpret the `TZ` environment variable.

Second, in both `clx_us`'s and `postgres`'s `~/.profile` you should set the `TZ` variable as follows:

```
TZ=GMT
export TZ
```

To change the CMOS clock from within Linux, you must use the command `/sbin/clock -w`. At boot time, to set the system clock from the CMOS clock, you should put `/sbin/clock -au` in your start files.

Also you should see that your computer' clock is not lagging or leading in time. There is a nice trick of keeping your clock accurate without the need for an external time base. This is done by first measuring the deviation over a certain period (say one week) and then specifying this deviation in the file `/etc/adjtime`. After that, the command `/sbin/clock -au` will automatically correct the hardware clock at regular intervals. The more exact the deviation is known, the better. Look into the man page for `/sbin/clock` (try `man 8 clock`) for exact details.

5.2 Log files and Syslogd

CLX makes extensive use of the `syslog` facility. This requires `syslogd` to run. You may redirect CLX's output to different log files. This is done in the `/etc/syslog.conf` file. There are several levels of logging output. Here is part of DB0CLX's `/etc/syslog.conf` file. The debug option generates lots of output, beware! Also with lots of traffic, the debug option generates so much traffic that the system may lock up. This is something we have observed at DB0BCC some time ago. When we discontinued the debug log, all was OK again.

```
local1,local2,local3,local4,local5.err          -/usr/local/clx/log/err.log
local1,local2,local3,local4,local5.crit         -/usr/local/clx/log/crit.log
local1,local2,local3,local4,local5.info         -/usr/local/clx/log/io.log
# local1,local2,local3,local4,local5.debug      -/usr/local/clx/log/debug.log
```

There is an important detail when specifying the file name: Normally, the `syslogd` syncs the file system (fsync) with every single message. This produces a very high system load especially when a lot of log messages ocur. With "-" in front of the filename, no fsyncs are triggered and the default Unix behaviour is used. This information is described in the syslog.conf manual page (try `man 5 syslog.conf`). We recommend to use "-", otherwise your system may become very slow.

When you make changes to `/etc/syslog.conf`, you will have to restart `syslogd` with `kill -1` to make it read the new version of the file. You can deactivate entries in the config file by putting a "`#`" in front of the commands.

Carl Makin, VK1KCM, writes that he feeds all log output into a single file with the following configuration:

```
local1,local2,local3,local4,local5.*     /usr/local/clx/log/all.log
```

5.3 Keeping track of CLX's Status

The CLX startup and shutdown script `~/bin/clx` also allows to check CLX's running status, i.e. if all is well. For this, you must call `clx` with the "`-c`" option.

```
$ clx -c
Checking clx processes...

Shared Memory manager               clx_ctl:    rc = OK
Internal Communications Manager     int_com:    rc = OK
Transmit Spooler                    snd_ctl:    rc = OK
User Administration                 usr_ctl:    rc = OK
Inter-User communication             iu_com:    rc = OK
Inter-Node communication            icl_com:    rc = OK
User Database Interface             usr_req:    rc = OK
User Data Table Manager             udt_mng:    rc = OK
Mailbox Controller                   mb_ctl:    rc = OK
User Commands Manager               usc_mng:    rc = OK
User Dialog manager                 usr_dlg:    rc = OK
BBS interface                        bbs_if:    rc = OK
Received messages Dispatcher        rm_disp:    rc = OK
Received messages Spooler           rcv_ctl:    rc = OK
AX.25 interface                     con_ctl:    rc = OK
$
```

Normally, all processes should return OK as their status. If one of the processes is missing, you should shutdown CLX and restart.

If this happens often, there maybe a problem with CLX, with the database or else. Please read the bugs chapter for known problems and errors.

Also `clx`'s return status can be used to determine a problem. CLX's own watchdog feature uses an internal function to check its status. For more information see 5.6 (CLX Watchdog).

5.4 Automatically starting CLX

For automatic startup you should make a link from `/sbin/init.d/rc3.d` or `/sbin/init.d/rc2.d` (what ever your default runlevel) to `/usr/local/clx/tools/startup`. This will then run the appropriate startup script when Unix is starting up.

```
# cd /sbin/init.d/rc{2|3}.d
# ln -s  ~clx_us/tools/startup S99clx
```

The startup file will remove any lock files and then start up CLX.

5.5 Shutting down CLX

Contrary to the original DOS-based Pavillion software, CLX does not need to be restarted at regular intervals due to memory leakage etc. Of course there may also be bugs in the software and memory leaks, but the impact is not so dramatic and so, in general, CLX can run for a long period without a restart.

If you still decide to shut down CLX regularly or even reboot your computer, you could use the following lines for root's crontab entry:

```
# crontab -l
1 0 * * * /usr/local/bin/adjtime
0 2 * * * /usr/local/clx/tools/clean_log
30 2 * * * /usr/local/clx/bin/clx -s
```

```
35 2 * * * /sbin/init.d/postgres stop
45 2 * * * /sbin/reboot
```

Read this as follows:

- CMOS clock is adjusted at 00:01 every day.
- At 02:00 the log files are cleaned up.
- At 02:30 CLX shuts down
- At 02:35 Postgres shuts down
- after a grace period of 10 minutes, the computer will be rebooted.

5.6 The CLX watchdog

As CLX may be instable or flakey at times, you may wish to control its status at regular times. With release 3.03, a watchdog concept was implemented into the core CLX software in that all the processes update a time stamp in the shared memory area once every sixty seconds. If all processes have timestamped that memory location, a file named `clx_stat` in the `~/log` directory is being updated. A script called `clx_watchdog` is called by crontab every 10 minutes and checks the status of this file. If the file has changed in the last five minutes all is left alone. However, if it hasn't, `clx_watchdog` shuts down CLX and restarts it.

To use the watchdog utility, put a line like this into your root's crontab file:

```
0,10,20,30,40,50 * * * * /usr/local/clx/tools/clx_watchdog
```

The watchdog script has undergone several changes and enhancements in the past. There is now a feature to turn off or turn back on the watchdog from the shell prompt and from inside the CLX program (as a privileged user only). The commands to control this are:

```
            | Shell as clx_us   | From within CLX (sysop mode)
-------------------------------------------------------------
Turn on     | clx_watchdog on   | enable/watchdog
Turn off    | clx_watchdog off  | disable/watchdog
Show status | clx_watchdog show | show/watchdog
-------------------------------------------------------------
```

This feature comes in handy when updating the CLX software or doing other time-consuming jobs when watchdog could interfere and shut down CLX when it was not really necessary.

5.7 Kernel Panic

A hint regarding this rare kernel feature. This portion is an excerpt from the Linux BootPrompt-HOWTO:

```
In the unlikely event of a kernel panic (i.e. an internal error that
has been detected by the kernel, and which the kernel decides is
serious enough to moan loudly and then halt everything), the default
behaviour is to just sit there until someone comes along and notices
the panic message on the screen and reboots the machine.  However if a
machine is running unattended in an isolated location it may be
desirable for it to automatically reset itself so that the machine
```

```
comes back on line. For example, using 'panic=30' at boot would cause
the kernel to try and reboot itself 30 seconds after the kernel panic
happened. A value of zero gives the default behaviour, which is to
wait forever.

Note that this timeout value can also be read and set via the
/proc/sys/kernel/panic sysctl interface.
```

5.8 Other regular Tasks

There are some other things which should be started regularly. For this, we have added sample file `crontab.clx_us` file in the `~/config` directory which looks like this:

```
# At 00:02 UTC every day, create a new MOTD file
# ---> This one must be adapted to your local needs
# ---> 2 0 * * * /usr/local/clx/tools/mk_motd
# At 00:11 UTC every day start database maintenance
11 0 * * * /usr/local/clx/tools/db_maint batch
```

This file can be activated with the following command:

```
$ crontab ~/config/crontab.clx_us
```

Whenever you make changes to this file, you must re-read it with the `crontab` command or you may directly edit the crontab entry with `crontab -e`.

5.9 The ~/tools directory

There are several scripts in the `~/tools` directory which may be used by the system administrator.

5.9.1 Database administration

```
bup_db          backup or restore all CLX databases.
```

When called with "`-s`" the utility will make an ASCII backup of all CLX database tables to the directory `~/backup` plus tar all files in the `~/box` directory, which represent any mail and bulletin files. With "`-r`" this program will read back the ASCII data into an empty, newly created database and unpack the tar file. `bup_db` will also back up user defined tables and additionally send a warning message if the formats of the old and new tables don't match. The strategy is as follows:

When backing up, `bup_db` saves any CLX table which is found in Postgres table `pg_class` except `ar_data` and `sys_dat` as these may change in a future version of CLX. When restoring data, `bup_db` restores everything from the `~/backup` directory except files ending in `*.tmpl`. This way it is possible to also restore user defined tables.

```
clx_db          destroy and create CLX Postgres tables
clx_idx         create indexes for the CLX Postgres tables
```

These programs are used to either restart the database in case it has become corrupt or for other reasons. You should not use them in a normal situation.

The db_maint program is used for many routine maintenance jobs. It can be called from within CLX (as a privileged user), from the command line or via cron in batch mode. For example, to cleanup the DX database, purge the user log, delete old user records from the database, delete old mail messages and finally run vacuum you could use the following crontab entry:

```
# at 03:08 every day, cleanup CLX database and run vacuum
8 3 * * * /usr/local/clx/tools/db_maint batch
```

For more details on how to use this program, see 5.10.4 (db_maint).

5.9.2 CLX Maintenance, Startup and Shutdown

```
startup         startup CLX with correct environment (.profile)
clean_log       compress yesterday's logs and start a new log today
clx_watchdog    check if all CLX processes are still alive and if
                not, shutdown and reboot
mk_motd         generate a new "Message of the Day"
```

You may have read about these programs before.

5.9.3 Other programs

```
mk_th           create some directories for mailbox (used within the clx script)
log_monitor     display excerpts from the syslog
us_adm          user administration tool (modify permission flags)
shmd            display shared memory contents
mbx_chk         a tool to check the consistency of entities under ~/box
rm_fwd          a script for cleaning up mail forwarding queues
check_adv_txt   a utility to help you translate the <tt/adv_txt/ file
read_ak1a       a program to import PacketCluster WWV/DX/OPERNAM/QSL/FUL files
filesend        program to send a file as a CLX mail message
d2asc           Converts dumps of the DX table to nicely formatted ASCII
do_fwd          cleaning up mail in the local mailbox and in the forward queues
```

A few more details about these tools:

us_adm

see section 6.3 (User Commands) for more details.

mbx_chk

A tool was created to keep CLX's database of files in sync with the entries under `~/box`. `mbx_chk` checks if all files have corresponding entries in the `ml_file` table and if all entries in the table have corresponding files under `~/box`. The remaining files/entries will be deleted. Also files with a negative size in the database (this is when the files are listed with "`#`" as the size symbol) will be deleted. When do negative file sizes ocur? This is when CLX is shut down during message forwarding.

`mbx_chk` may be run at any time, with CLX up or down.

rm_fwd

This script is useful when you wish to clean up your mail forwarding queue to a specific node. Let's say, node `yy0yy` has ceased to exist but there is still a bunch of mail to be forwarded to him in your `iclb` directory. To remove these mails (which can never be sucessfully forwarded anyway), use the command `rm_fwd yy0yy`. You may also use this command from within the db_maint utility.

do_fwd

do_fwd is used to clean up the local mailbox and the forwarding queue. The strategy is basically to honour any changes in the user home node table and apply these changes accordingly:

- if a user has changed his home node to a different node, his unread mail is removed from the local mailbox and being put on the forward queue for that system. If he was a remote user and has changed to an yet other node, his mail from the forwarding queue for the previous node is now put on the new node.
- if a user has changed his home node to our system, any mail that's still stuck in the forwarding queue for his old home node is moved to the local mailbox.

shmd

A utililty to dump shared memory contents. This program is mainly for debugging purposes so that we can ask you to mail us a shared memory dump in case you are observing a strange error.

log_monitor

A little program to follow the log messages which are kind of hard to read. This script can filter out specific callsigns. Use `log_monitor -h` to find out its options. Basically they are:

```
-f      to focus on a specific callsign. Only messages to and
        from that call are being shown.

-x      to exclude messages to or from a specific callsign.

-w      to adjust screen width of the messages. Your screen looks
        bad when the messages shown are wrapping around the right
        end of the terminal and continue on the next. To change this,
        you can limit the number of characters shown to let's say
        80 and so any longer lines will be truncated.

-s      Starting time. The time must be given in a format like
        ``Jul 19 08:20:30''. This is for off-line monitoring of
        old log files. See decription below.
```

Here is a sample session to show you what's possible:

```
$ log_monitor -f dl6rai
CLX log monitor v1.0 -- Focus: "dl6rai"
dl6rai->: Connect.
dl6rai->: *** connected to dl6rai
dl6rai<-: Hi Ben, here is "clx"! Experimental cluster-software on linux.
dl6rai<-: clx >
dl6rai->:
dl6rai->:
dl6rai<-: clx >
dl6rai->: sh/us
dl6rai->:
dl6rai<-: User:
dl6rai<-:       dl6rai
dl6rai<-: clx >
dl6rai->: sh/dx/2 1
dl6rai->:
dl6rai<-:   1835.7  TI4CF        6-Oct-1996 0641Z                    <dl8ui>
dl6rai<-:   1842.9  D44BC        6-Oct-1996 0528Z                    <dl1yd>
```

```
dl6rai<-: clx >
dl6rai->: bye
dl6rai->:
dl6rai<-: Bye...
dl6rai<-: Disconnect.
```

This tool may also be called via the privileged command `monitor` when you are logged in as a CLX admin or superuser. You just specify the callsign to be monitored on the command line:

```
monitor dl6rai
```

Using command `f` you can direct the focus to another callsign, and with `q` you can leave the monitor. By default, the window size shown is limited to 65 characters to make it look nice on a 80x24 terminal. This parameter can be changed with the `w` command from inside the monitor.

When specifying a file name, you can trace an existing log file (off-line monitoring). The log file will then be traced from the begining unless you specify a starting time using the "-s"option. This allows you to watch a specific situation to find problems or misbehaviours. When "-s" was specified, the log is printed in quasi-realtime, and the time given in square brackets shows when the net log entry is to be expected. Try this out if it interests you and you will quickly understand what's going on.

check_adv_txt

a tool to check translated `adv_txt` files for completeness and generally helping you with translation. Please refer to section 3.5 (check_adv_txt) for further details.

read_ak1a

This utility is provided to import data from the original PacketCluster files `DX.DAT`, `OPERNAM.DAT` and `WWV.DAT` and QSL information from a PacketCluster `.ful` file. Additionally, DL8EBW's VHF database can be read using the `VHF` type.

The program has several flags and options:

```
-f              Output to a file named "postgres.input". Normally
                read_ak1a directly connects to the postmaster and
                feeds the data into the table.
-v              Be verbose
-p              Generate PSQL statements instead of "COPY from"
-c              Create a new table. Normally, read_ak1a appends data
                to an existing table.
-t <type>       Here you must specify the type of data you are
                reading in. This is either DX or OPERNAM, WWV, QSL,
                FUL, VHF or IOTA.
-r <comment>    Add a comment to the table being created. This feature
                is only available when type=FUL and is being used for
                creating new Udt-tables.
-k <rights>     Defines with which access rights the Udt table is being
                created. Default is "+-" (read-only). This feature
                is only available when type=FUL.
```

`read_ak1a` transfers AK1A PacketCluster data to the following CLX tables:

```
DX.DAT          dx_data
OPERNAM.DAT     us_data and us_uhn
WWV.DAT         wwv_data
QSL.FUL         qsl_mng
*.FUL           any other Udt table
```

Using the program is very straightforward. Let's say, you wish to import your old PacketCluster files one by one into clx. Here are the steps you have to do.

1. copy the files `DX.DAT`, `OPERNAM.DAT` and `WWV.DAT` and some `*.ful` files from PacketCluster into a temporary directory, say `~clx_us/tmp`:
2. run the following import commands:

```
$ read_ak1a -t dx dx.dat
$ read_ak1a -t opernam opernam.dat
$ read_ak1a -t wwv wwv.dat
$ read_ak1a -t qsl qsl.ful
$ read_ak1a -t ful address.ful address
```

   With the last two commands you must also specify which Udt table this data is to be applied to. If you wish to create a new table you must also specify the `-c` flag and you may optionally add more flags for the Udt tables being created:

```
$ read_ak1a -c -t dx dx.dat
$ read_ak1a -c -t opernam opernam.dat
$ read_ak1a -c -t wwv wwv.dat
$ read_ak1a -c -t qsl qsl.ful
$ read_ak1a -c -r "Address Database" -k "++" -t ful address.ful address
```

If you encounter any trouble and see syntax error messages, this indicates a problem with the original data. The files I got from DB0BCC were partly corrupted, especially the `OPERNAM.DAT` contained a lot of trash and I could only use a small part of it. To find out where the trouble is, first create an intermediate file from the data and take a look at it. This is what I did with my `OPERNAM.DAT`:

```
$ read_ak1a -f -t opernam opernam.dat
$ vi postgres.input
$ dd if=opernam.dat bs=196 count=121 of=opernam.new
$ read_ak1a -c -t opernam opernam.new
```

First I converted the complete file into the intermediate file `postgres.input` and had a look at it. I decided that only the first 121 records (who are 196 bytes long each) were useful. The rest of the data was completely trashed. With the `dd` command, I extracted the first 121 records and wrote them to a new file `opernam.new`. Finally I imported this file into my data table.

Using this program is not a prerequisite to run CLX. It just provides a way to save your long-time DX information to the new system or import some available database files into CLX. If you decide to completely start from scratch, just go ahead.

Here are a few benchmarks reading in DX.DAT files run on a 90 MHz Pentium machine with 64 MB RAM:

```
 Reading 10,000 DX spots:     46 seconds
Reading 100,000 DX spots:    648 seconds
Reading 486,333 DX spots:   7911 seconds
```

5.10 Admin Commands

There are a few commands which are reserved for the CLX admin. Any user who has the "`admin`" bit set in his user data record is treated as a CLX admin. Additionally, any user who has successfully passed the `set/privi` or the `pw` dialog, does have admin privileges.

5.10.1 Achieving Admin Status

There are four ways to achieve admin status:

1. automatically by modifying the admin flag with `us_adm`. This way, a specific callsign is always privileged. This is for relatively private installations where there is no fear of radio pirates.

2. manually by using `set/priv` and supplying a password. This password must be generated using the `~/bin/get_pwd` program. It is a string which is calculated from a random number and the `pw:` entry in the `~/config/clx_par` file. The dialog that follows `set/priv`, CLX presents you with a code:

   ```
   set/priv
   Get password for: 866992064
   ```

   Now you must call the program `~/bin/get_pwd`. This implies that you have Linux running locally because `~/bin/get_pwd` is a statically linked ELF binary and cannot be run under MS-DOS or Windows.

   ```
   $ ~/bin/get_pwd abcdefgh 866992064
   ^BTg]V>FUUF
   $
   ```

   Now return the password back to CLX.

   ```
   set/priv
   Get password for: 866992064
   ^BTg]V>FUUF
   Tnx.
   dl6rai-3 de xx0xx   17-Jul-1997 1752Z   clx >
   ```

   CLX will either respond with `Tnx.` or with `Sri.` depending on whether you replied with the correct password or not. Un-setting the privileges is done with the command `set/nopriv`. Users with a permanent admin status in the user record cannot switch off their privileged status in an way.

   See 3.1 (CLX parameters in ~/config/clx_par) how to specify the `pw:` field in the CLX configuration file.

3. manually by using the `pw` command. This is compatible with THENET or BAYCOM password generation. For this you must specify a password in the `~/config/clx_par` file using the `baycom_pw` command. The minimum length of the baycom password is 5 characters.

   ```
   dl6rai-3 de xx0xx   17-Jul-1997 1745Z   clx >
   pw
   DB0CLX>  29 11 35 9 8
   f5&94
   dl6rai-3 de xx0xx   17-Jul-1997 1745Z   clx >
   ```

   You may also add digital noise to the password by putting a random number of arbitrary characters in front and behind the password:

   ```
   35g0al54w4zhagltkf5&94tr904w6zhtskarel94w
                     ^^^^^
   ```

   To get back to unprivileged status use the command `pw off`. Users with a permanent admin status in the user record cannot switch off their privileged status in an way.

   See 3.1 (CLX parameters in ~/config/clx_par) how to specify the `baycom_pw:` field in the CLX configuration file.

4. through authorization from a CLX super user. A super user may use the command `set/priv <call>` to change the user status for `<call>` to "admin". This status is kept for the current connection only, it is not permanent. Admin status can be removed again by using `set/nopriv <call>`.

5.10.2 Admin shell commands

Generally, the CLX admin can use any program located under the `~/exec/privileg` directory. By default, there are a few commands like `!`, which allows executing Unix commands from the clx prompt:

```
! pwd
/usr/local/clx
dl6rai de db0clx   20-Jul-1997 0807Z   clx >
```

These commands are executed with standard `clx_us` privileges.

Also there is an interactive `shell` command which allows you to spawn a shell and interactively work with the command prompt. the `shell` and `!` are in fact identical, the `!` is just a Unix convention found in many programs. However, note the blank which must go between the `!` and the command.

```
dl6rai de db0clx   20-Jul-1997 0808Z   clx >
shell
$ cd config
$ ed cluster_par
301
/bcc
-db0bcc         xa+     tnt-router
s/-db0bcc/db0bcc/
w
300
q
$ exit
exit
dl6rai de db0clx   20-Jul-1997 0810Z   clx >
```

The above is an example how you can edit the `~/config/cluster_par` file from within CLX. You may even call the editor from the CLX command line:

```
dl6rai-2 de db0clx   23-Nov-1997 1834Z   clx >
! ed config/cluster
301
q
dl6rai-2 de db0clx   23-Nov-1997 1834Z   clx >
```

You can use other programs this way too the like:

```
! clx -c
```

There are a few other programs/scripts in that directory like `ps` to list process status from within the CLX shell, or a `ping` command to generate a `PC51`, or `mon` which lets you call the `log_monitor` described in 5.9.3 (Log Monitor).

5.10.3 Admin commands for the ~/box directory

There are some basic commands for the CLX admin to modify any files under the `~/box` subdirectory:

```
ls      to list a directory relative to the ~/box directory
get     to read a file
put     to write/create a file
rm      to erase a file
mkdir   to create a new directory under ~/box
```

These commands will basically do two things:

1. create/modify/delete a file in the specified directory or create a new directory
2. modify the database accordingly

With this you may create a new file area or change the Message of the day. With the following CLX command you can enter a new "Message of the Day" message:

```
put info/help/motd
```

Please refer to the section 3.7 (MOTD) describing how to set the message of the day automatically.

Or with this command you can examine DJ0ZY's login script:

```
get batch/start/dj0zy
```

5.10.4 User Data Table Commands (Udt)

User Data Tables are general purpose database tables for storing information like addresses, IOTA information etc. These are the commands to administer these tables.

UDT Admin Commands

```
create/udt <tablename>/<flags> <cmnt>   create a new table
destroy/udt <tablename>                 destroy an existing table
info/udt <tablename>                    show info about table
```

User commands

```
show/<tablename> <key>                  searches the key field and
                                        outputs matching values
                                        (exact match)
show/<tablename> ~<partial key>         searches the key field and
                                        outputs matching values
                                        (partial match)
delete/<tablename> <key>                delete a record
update/<tablename> <key>                update a record
```

The names of the tables may use characters `a-z`, underscore "`_`" and the minus sign "`-`". The length of the name is limited to 3-10 characters. Internally (i.e. in Postgres) the table is created with the prefix "`Udt_`". The structure of the table is very simple and essentially identical to PacketCluster's "`.ful`" style files. The keyword must consist of characters `a-z`, underscore "`_`" and the minus sign "`-`" and may be 1-16 characters in length. The search is case-insensitive.

The content field may contain roughly 8000 characters, so it may be a single line or several paragraphs long.

The information record, which is created automatically when the table is created from within CLX, may contain a comment of up to 255 characters.

There are flags for `user write` and `user read`. These flags are specified when the table is created with `+` and `-` characters. The first, leftmost digit is the read symbol, the second one is the write symbol.

```
++      User may read and write the table
+-      User may read the table only (this is the default)
-+      User may write the table (but not read --- does this make sense?)
--      User may not read nor write the table
```

The information about the table (user rights, comment etc.) is contained in the `Clx_Udt_Info` record. This record can not be accessed directly by the user as all user supplied keywords are converted to lower case before passing them on to Postgres.

The CLX admin may create or destroy Udt tables, as described in section 5.10.3 (User Data Table Commands). Please refer to this section for further details.

5.10.5 Database Maintenance Tool

In order to maintain the CLX database, remove old or erroneous entries, delete old mail messages and other such tasks, a little menu-driven tool was put under the privileged path. Any user with sysop status can call up this menu. The output of the program is very limited:

```
db_maint
(db_maint:dl6rai) -->
```

The program stops and waits for your input. Using the `h` command, you may list the available options:

```
db_maint
(db_maint:dl6rai) --> h

CLX Database maintenance program, main menu

 1 -- DX Table related functions
 2 -- WWV Table related functions
 3 -- Statistical functions
 4 -- Modify user records
 5 -- User log related functions
 6 -- Mailbox related functions
 7 -- QSL table related functions
 8 -- Other database table functions
 9 -- General functions
 q -- Exit

(db_maint:clx_us) --> _
```

The available options are pretty self-explaining. The `db_maint` program is also available from the `tools` directory and can be run by cron to do periodic maintenance jobs, like deleting old mail messages, purging log records etc. For this to work well, `db_maint` silently turns off the CLX watchdog and back on after it has finished.

Several parameters for `db_maint` can be specified in the `clx_par` file like aging of user records or mail messages.

For running db_maint via cron, a special mode, the "batch" mode was implemented which executes a number of commands which can be specified in the `clx_par` file with the `batchcommands:` parameter. Other parameters can also be set there like, for how many days log entries should be kept, how long user records should be kept, how many days back DX spots should be kept etc. etc.

For further details see 3.1 (CLX parameters in ~/config/clx_par).

5.10.6 Managing Distribution Lists

With CLX version 3.03 and later, distribution lists are available. These are a kind of symbolic addresses which can be used both with mail and announce commands by the users. This is useful for sending special interest bulletins to a group of people as a private mail.

Basically there are three commands to manage the distribution lists:

1. `set/distro` to add a user to a list - a new list will automatically be created.
2. `set/nodistro` to remove a user from a distribution list - if the iist is empty, it will automatically be deleted.
3. `show/distro` to find out who is on a specific distribution list or which lists are available.

Here is an example: Let's say we wish to create a new distribution list called `SIX` for the 6 meter enthusiasts. So here we go:

```
set/distro six dl7av
set/distro six dj5mn
set/distro six dj1oj
```

Now DL7AV, DJ5MN and DJ1OJ are on the list as we can see here:

```
sh/distro six
        dj1oj        dj5mn        dl7av
dl6rai de xx0xx   31-Jan-1998 2051Z   clx >
```

To see which other lists are available, we can use the following command:

```
sh/distro
          six          ukw
dl6rai de xx0xx   31-Jan-1998 2051Z   clx >
```

One other list named `ukw` is available too. To remove DJ5MN from the SIX list, we use the following command:

```
set/nodistro ukw dj5mn
```

Thats all about distribution lists. Users can send messages to these lists using the symbolic names instead of callsings with the `send` command. Additionally, they may use the `announce` command to make directed announcements to a specific distibution list.

5.10.7 Checking for bad words in mail

In some countries, like the U.K. the sysop is responsible for the messages on his system. He is required to have a means to auomatically check for mail messages containing evil words or expressions. To make life easier, CLX allows creating a file where you can specify bad words. A script in the special directory `~clx_us/exec/checks` is called for every incoming mail and if the message contains a bad word, it is disregarded. Simply list the bad words one by one (each in a new line) in the `~/config/bad_words` file.

Sample bad words file:

```
milk
alcohol
```

This bad_words file will kill all incoming messages which contain the words `milk` or `alcohol`.

5.11 Superuser Status

The difference between a CLX `admin` and `superuser` is that only the superuser may give `admin` rights to other users. This is done with "`set/priv <call>`". (See also 3 (Achieving Admin Status)). To become a superuser, you must use the procedures described in section 5.11 (User Administration).

The first user who logs in after installing the CLX software (usually from the console using the `net_usr` command) will automatically be granted superuser rights. Note that logins with `net_usr` automatically have SSID -16 if nothing else is specified.

5.12 Extending CLX – the ~/exec/command directory

You may put your own command extensions into this directory. Anything found here is executable by the user. CLX (in fact the program `usc_mng`) calls the program with at least one parameter, namely the user's callsign. If the user specifies anything on the command line, these parameters are passed as `$2`, `$3`, etc.

As an example for extensions, we provide the program `sun` in that directory which calculates sunrise and sunset times. When DL6RAI enters

```
sh/sun kl7
```

on the command line, the program is called from `usc_mng` as follows:

```
~clx_us/exec/command/sh/sun dl6rai kl7
```

The sources of the program `sun` are available as an example application. You may create your own and if you like, you may send them to us for inclusion in future releases of the CLX software.

5.13 Extending CLX even further – the ~/exec/interpr directory

After a while experimenting with the external commands, it became clear that to seamlessly integrate external programs into CLX, one would need to achieve national language support too. However, adding this functionality into the external program seemed very complicated and unwise, as CLX already provides this functionality. So we took another approach.

A new directory was added to the list of command extension directories under `~/exec`, called `interpr`. Programs in this directory are bound to produce a slightly different output in the form:

```
\<msg-nr>\tab<par1>\tab<par2>\tab .... \tab<parn>\n
```

The first parameter is the message number requested from **adv_txt**. this is a number of 1-4 figures prefixed with a backslash. This may be followed by one or more (max. 20) parameters which are filled into the appropriate "%s" fields in the message. The parameters must be separated by TABs and can only be strings, no binary data. The last parameter must end with a linefeed. The message numbers must be between 001 and 2000. Missing parameters are filled with the empty string.

CLX now takes this output and interprets it in the appropriate language and fills in the missing variables in the output string.

5.14 The interactive clx_adm tool

This tool gives you the ability to view the status of the node, and start or stop connections to other nodes or users. In the future, some of the features currently provided by external tools will be integrated into **clx_adm**.

clx_adm comes up with an interactive surface:

```
#=== C L X ===#
[ Function:   [
#=============#
```

You may now press <F2> to see the menu of available sub menus:

```
#=== C L X ===#
[ Function:   [
#==========+-----------+
           | b basics  |
           | c cluster |
           | u user    |
           +-----------+
```

Ian Maude, G0VGS, reports that he had to set an environment variable called **TERMINFO** to make this look correctly. What he did was

```
$ export TERMINFO=/usr/share/terminfo
```

before starting **clx_adm**.

Select a sub menu with the cursor-up/cursor-down keys or by pressing the first characters. Then press <Enter>. Selecting the basics-Menu, for example, gives you a status report of the system currently running:

```
#=== C L X ===#
[ Function: b [
#=+-Basics-----------------------------------------------------+
  |                                                           |
  | Version:  4.02    emulate: 5447                           |
  |                                                           |
  | Call: xx0xx          Last Start-Up: 20-Sep-1998 0734Z     |
  |                                                           |
  | Data-Base: on    Name: clx_db      Host:                  |
  |                                                           |
  +-----------------------------------------------------------+
```

Pressing <F4> will bring up a small list to select from:

```
#=== C L X ===#
[ Function: b [
#=+-Basics-------------------------------------------------+
  |                                                        |
  | Version:  4.02       emulate: 5447                     |
  |       +-------------+                                  |
  | Call: | clx_par     |Last Start-Up: 20-Sep-1998 0734Z  |
  |       | cluster_par |                                  |
  | Data-B| adv_txt     |: clx_db      Host:               |
  |       +-------------+                                  |
  +--------------------------------------------------------+
```

This is the select list to make CLX re-read one of the two configuration files or the language dependend message files. If this action is not triggered manually, CLX will automatically notice the change of the configuration files within the next five minutes.

One can also use this feature from the command line:

```
$ clx_adm +b clx_par
```

This command will also make CLX re-read the configuration file **clx_par**.

Returning to the previous menu level is possible by pressing <ESC> twice. So <ESC><ESC> brings you back to the main menu.

The second sub menu cluster allows you to instantly lock/unlock, connect and disconnect nodes in your **~/config/cluster_par** file. You cannot add any entries to the file here but you can force a connect. This is a nice option for testing out new routes or connect scripts.

Press '**c**' or select the sub menu with the cursor-up/cursor-down keys.

```
#=== C L X ===#
[ Function: c [
#=+-Cluster----------+
  |                  |
  | Call:            |
  |                  |
  +------------------+
```

You will then be prompted for a callsign. You may now either enter one callsign from the **cluster_par** file directly or rather press <Enter>. Now you will see the list of all currently known nodes for selection:

```
#=== C L X ===#
[ Function: c [
#=+-Cluster----------+
  |                  |
  | Call:            |
  |     +-------------------------------------------------------+
  +-----| db0clx      [xx0xx       ]   >cl       act   clx  PC  a  |
        | monitor     [xx0xx       ]   >cl    ^  pass  clx  U      |
        | db0bcc      [xx0xx       ]   >cl       act   clx  PC  a  |
        |                                                         |
        |                                                         |
```

This screen is actually a status report of the current link situation. It shows (in this order):

1. The link partner's callsign.
2. The callsign that is used for the AX.25 connection (incoming or outgoing).
3. A flag showing if the link is locked. Either "`m`" (manual lock), "`f`" (locked by cluster_par), "`x`" (both locks) or "" (no lock).
4. A flag showing if the link is outgoing (>) or incoming (<) and if the partner is a cluster (`cl`). This later flag will be deleted in a future version as it does not make much sense any more.
5. Link status like it is shown in `SHOW/CONFIGURATION`.
6. A flag showing the link status being active (`act`) or passive (`pass`).
7. A flag showing if the partner is CLX or non-CLX (`clx` or `-`).
8. A flag showing which protocol is used: PCxx protocol (`PC`) or user mode protocol (`U`).
9. A flag showing the interface type used (`a/w/x/i`).
10. The ping timer.
11. The ping timout limit.

You can now select one of them by pressing the <Enter> key. Now press <F4>. A list of available commands is displayed:

```
#=== C L X ===#
[ Function: c [
#=+-Cluster----------+
  |                  |
  | Call: db0bcc     |
  |       +------------+
  +-------| lock       |
          | unlock     |
          | connect    |
          | disconnect |
          | ping       |
          +------------+
```

Locking means that you can temporarily lock a node callsign from being automatically connected. The `unlock` function temporarily also unlocks a callsign from the locked list. No permanent changes are applied to the `cluster_par` file, after a restart of CLX, all changes made by `lock/unlock` are gone.

You can now select one of these options or simply go back with <ESC><ESC> to the previous menu.

Finally, the third sub menu is designed for user callsigns. It looks very similar to the previous one:

```
#=== C L X ===#
[ Function: u [
#=+-User-------------+
  |                  |
  | Call:            |
  |                  |
  +------------------+
```

You may now either enter a user's callsign (who must be currently connected) or simply press <Enter> to select one from the list of users currently connected. Then, pressing <F4> brings up a choice of options available.

```
#=== C L X ===#
[ Function: u [
#=+-User-------------+
  |                  |
  | Call: xx0xx      |
  |       +----------+
  +-------| disconnect |
          | log        |
          +------------+
```

So this is it at the moment. You may also use **clx_adm** in a non-interactive way by simply stating the commands on the command line. For example:

```
$ clx_adm +c connect db0bcc
```

will make CLX instantly startup the connection to DB0BCC. Or,

```
$ clx_adm +u disconnect df2rg
```

will instantly disconnect DF2RG.

6 User Administration

6.1 User Commands

User commands are now available as a user manual contributed by Ian, G0VGS. You will find it in the directory **~clx_us/doc/user**.

6.2 User flags

Users can have different attributes and rights. These user flags are stored together with other user specific information like name, QTH, page length etc. in the user database. For every callsign there is exactly one database entry.

As in AX.25 users can show up with up to 16 different SSIDs (-0...-15) one set of flags is stored for every SSID. Additionally, there is one more SSID (-16) for logging in from the system console.

The following flags exist for users. These flags may either be specified for a specifiy SSID or (if the speicific SSID flag is -1) as a default for all SSIDs.

```
flag    attribute       meaning

-1      default         user has default permissions
1       login_denied    user will be disconnected
2       priority_login  user can login even if generally not permitted
4       login_ignored   login is ignored (i.e. digipeater)
8       admin           user is admin
16      superuser       user is superuser
```

The `login_denied` attribute is equivalent to PacketCluster's `SET/LOCKOUT` command. The user will be disconnected immediately after his connect without any warning or informal message.

The `priority_login` flag is reserved for special cases, where login is generally not permitted. Normal users are then graciously disconnected after being informed with a message saying that the node is not available at this time. Users with the `priority_login` flag set will still be able to connect.

With the Flexnet type of net nodes making a connect every 10 minutes or so, it was felt necessary to add the attribute `login_ignored`. The station is logged in but — as it is generally not interested in receiving any information but rather checking the link connection and measuring the time — the login will be ignored, i.e. nothing will be written to the user database and no greeting message is produced by CLX for such users.

The `admin` flag is a permission for users who have special tasks to do. see section 5.10 (admin) for more details.

The `superuser` flag gives a user even more rights. see section 5.11 (superuser) for more details.

The `default` flag has a special meaning giving the user the default rights as defined in the default permission flag.

With these flags it is possible to easily administer users and give special attributes to some. Normal users will have -1 as their SSID permission flag and 0 as their default permission. That means, they have no special rights.

Some examples: To exclude a "`NOCALL`" (an often-heard pirate call) with any SSID, put the default permission to `1` and leave the ssid permissions at `-1`. To give `admin` and `priority_login` rights to DL6RAI-13, change the ssid permission to 10 for "DL6RAI-13" and leave all others at -1. To set the name of "DJ0ZY" to "Franta" you would use the "`-n`" option. To ignore any logins from (RMNC, Flexnet) digipeaters, let's say for DB0PV, put the default permission to `4`. This will make both Flexnet and CLX happy.

6.3 The us_adm tool

These flags and other attributes can be looked up and changed with the `us_adm` program in the `tools` directory. To follow the examples above you would issue the following commands:

```
$ us_adm -d 1 n0call
$ us_adm -p 10 dl6rai-13
$ us_adm -n Franta dj0zy
$ us_adm -n 4 db0pv
```

When you call `us_adm` with the `-v` flag and a callsign, the program lists the user's entries in the database:

```
$ us_adm -v dl6rai

User Information for Station dl6rai

                Name: Ben
```

```
           Location: 48 34 n 012 12 e
                QTH: Ergolding
         First login: Sat Feb 03 09:14:08 1996 GMT
          Last login: Thu Feb 29 16:47:32 1996 GMT

          Pagelength: [default]
            Language: german
       Character Set:

         Exit string: /exit
             Address:

SSID-Permission Flags
-------------------------------------------------------
       SSID:  0  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15
Permissions: -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1
-------------------------------------------------------

Default Permission: 0    Flag for dl6rai-0: -1 [default]

$
```

With CLX version 2.03 `us_adm` was reworked and can now also create new users. So you it's not necessary to have a user log in before you can change his entries. Also the output is slightly different. You can also erase users with the "`-e`" switch (this was just added for symmetry). Just call the `us_adm` program without any parameters to see what you can do. When a field has blanks in it, you must surround it with double quotes ("`48 34 N 12 12 E`").

`set/home_node` allows the sysop updating a user's home node information. `set/home_node dl6rai db0bcc` changes DL6RAI's home node setting to DB0BCC.

6.4 Connecting, Disconnecting and Locking out

Privileged users may use the commands `disconnect`, `set/lockout` and `set/nolockout` from within CLX to either force a disconnect or even lock out the callsign completely. Disconnecting and locking out a station is quite a drastic measure and should be used only in an emergency. Both actions are logged to `io.log` for later investigation. With `set/nolockout` the ban on the station previously locked out is being removed.

There is also a utility called `monitor` which can be used to monitor user input and output. This is described in section 5.9.3 (log_monitor).

With the `connect` command you may trigger CLX to start up a link connection and as such avoid lengthy idle times when the CLX timers are running.

7 Appendix

7.1 Release Notes

The release notes are no longer in this manual. Please look for the files called `README_*` in the `~/doc/sysop` directory.

7.2 PacketCluster's PCxx protocol

The following list comes from the original PacketCluster documentation. These are the messages used among PacketCluster and CLX nodes. Thanks to OE1TKW for originally collecting this information:

7.2.1 Excerpt the from PacketCluster sysop manual, Appendix C

Internode communication between PacketCluster nodes is performed by sending Information packets that begin with the string "PC", followed by an ACSII coded number between 10 and (currently) 51.

7.2.2 Overview

```
PC10    Talk mode
PC11    DX info
PC12    Announcement
PC13    Stn into CONF
PC14    Stn out of CONF
PC15    Conference mode
PC16    PC stn add
PC17    PC stn delete
PC18    Initialisation: RequestInit
PC19    Initialisation: NodeAdd
PC20    Initialisation: InitDone
PC21    Initialisation: NodeDelete
PC22    Initialisation: PCDone
PC23    WWV Info
PC24    Here status info
PC25    DX/WWV merge request
PC26    DX merge info
PC27    WWV merge info
PC28    mail: SendSubject
PC29    mail: SendText
PC30    mail: AckSubject
PC31    mail: AckText
PC32    mail: CompleteText
PC33    mail: AckCompleteText
PC34    Remote commands: Command *
PC35    Remote commands: Response *
PC36    Remote commands: Show command
PC37    Needs database update
PC38    Connected node list
PC39    NodeDelete w/disc
PC40    PC file forward
PC41    User info
PC42    Forwarding abort
PC43    PC-mail: External mail, Send subject
PC44    Remote DB request
PC45    Remote DB response
PC46    Remote DB complete
PC47    Remote DB update
PC48    Remote user DB update
PC49    Bulletin mail delete
PC50    Local user count
PC51    Ping request or answer
```

```
*) PC34 and PC35 are PC84 and PC85 for CLX-to-CLX remote commands.
```

7.2.3 CLX additions to the PCxx protocol

The following command extensions are being used within the CLX network:

```
PC73    WCY Info
PC75    WCY merge request
PC77    WCY merge data
PC84    CLX remote command
PC85    CLX remote command response
```

7.2.4 Syntax description

```
PC10^from-stn^to-stn^msg^bell-flag^to-stn^from-pc^~
PC11^DXfreq^DXcall^DXmisc^Date^Time^logger^from-pc^hops^~
PC12^from-stn^to-PC^msg^sysop-flg^from-pc^wx-flg^hops^~
PC13^stn^hops^
PC14^stn^hops^
PC15^from-stn^msg^hops^
PC16^host^stn conf-mode here^stn conf-mode here^stn conf-mode here^...^...^hops^
PC17^stn^from-pc^hops^
PC18^cluster-info^ver^~
PC19^here^PC-stn^talk^version^hops^
PC20^
PC21^PC-stn^reason^hops^
PC22^
PC23^date^hour^SFI^A^K^forecast^logger^from-pc^hops^~
PC24^stn^here^hops^
PC25^merge-stn^from-stn^DX-cnt^WWV-cnt^
PC26^DXFreq^DXCall^date^time^info^logger^to-stn^~
PC27^date^hour^SFI^A^K^forecast^logger^to-stn^~
PC28^to-pc^from-pc^to-stn^from-stn^date^time^private-flag^subject^~
PC29^to-pc^from-pc^msg-#^text^~
PC30^to-pc^from-pc^msg-#^
PC31^to-pc^from-pc^msg-#^
PC32^to-pc^from-pc^msg-#^
PC33^to-pc^from-pc^msg-#^
PC34^to-pc^from-pc^cmd^~
PC35^to-pc^from-pc^cmd-resp^~
PC36^to-pc^from-pc^cmd^~
PC37^to-pc^from-pc^stream^command^~
PC38^node,node,node,...^~
PC39^stn^reason^
PC40^to-pc^from-pc^filename^bull-flag^line-cnt^
PC41^stn^type^info^hops^~
PC42^to-pc^from-pc^msg-#^
PC43^to-pc^from-pc^to-stn^date^time^private-flg^subject^line-cnt^~
PC44^to-pc^from-pc^stream^qualifier^key^user^
PC45^to-pc^from-pc^stream^info^~
PC46^to-pc^from-pc^stream^
PC47^to-pc^from-pc^user^qualifier^key^stream^type^
PC48^to-pc^from-pc^stream^qualifier^key^user^
```

```
PC49^stn^subject^hops^~
PC50^from-pc^usercnt^hops^
PC51^to-pc^from-pc^ping^

Legend
======

logger       station that reports the info
bell-flag    1=bell, 0=no bell
merge-stn    node that provides the info
DXCall       DX station call
msg          message text
DXFreq       DX station frequency
PC-stn       PacketCluster node
DXmisc       DX misc text
stn          Call of connected station
from-stn     originating station
conf-mode    *=conf, -= no conf
here         1=here, 0=no here
to-stn       destination station
host         Packet Cluster node call
ping         1=request, 0=answer
```

7.2.5 Protocol messages handled by the CLX software

This is a table of **PCxx** messages which are currently handled by the CLX software.

Msg	process	generate	forward	note
PC0	error	-	-	dj6rx
PC10	+	+	+	
PC11	+	+	+	
PC12	+	+	+	
PC13	-	-	-	
PC14	-	-	-	
PC15	-	-	-	
PC16	+	+	+	1)
PC17	+	+	+	1)
PC18	+		+	
PC19	+	+	+	1)
PC20	+		+	
PC21	+	+	+	1)
PC22	+		+	
PC23	+	+	+	
PC24	+	+	+	
PC25	+	+	+	
PC26	+	+	+	
PC27	+	+	+	
PC28	+	+	+	
PC29	+	+	+	
PC30	+	+	+	
PC31	+	+	+	after 9999 lines
PC32	+	+	+	
PC33	+	+	+	

```
PC34        -         -         -
PC35        -         -         -
PC36        -         -         -
PC37        -         -         -
PC38        -                   -
PC39        +         +         -
PC40        -         -         -
PC41        +         +         -
PC42        -         -         -
PC43        -         -         -
PC44        +         +         -
PC45        -         +         +
PC46        -         +         +
PC47        -         -         -
PC48        -         -         -
PC49        -         -         -
PC50        +         +         -
PC51        +         +         +
------------------------------------------------------------
```

Notes:

- `+` means CLX processes this request.
- `-` means CLX ignores this request.
- An empty field denotes meaningless fields.

The field `forward` has two meanings:

1. broadcast type messages (see 3.3 (Broadcast type PCxx telegrams)) will be forwarded to other nodes according to CLX's forwarding scheme (which knows active and passive link partners as well as CLX and non-CLX partners). If the hop counter in the received telegram is zero, the telegram will not be forwarded.
2. Telegrams with a destination (like mail forwarding or database request) will be forwarded normally to the destination.

`1)` means: Due to compatibility issues with the Pavillion software, this message is not being forwarded to a PacketCluster node.

7.3 Current Users of the CLX software

This list has grown too big. Please refer to the file `/usr/local/clx/doc/misc/user.db` for information about current users of the CLX software. Email addresses shown with an initial `#` did not work, so they are probably out of date.

7.4 Thank You!

Numerous people have helped out with information, details, bug reports and other useful things in the years of the CLX development. Most of our communication was carried out via the Internet, some by Packet Radio. Thanks to all of you for your help, especially to:

```
Alan Cox, GW4PTS for putting the AX.25 code into the kernel
Jonathan Naylor, G4KLX, for maintaining the AX.25 code
Harm, DG7DAH for helping us with WAMPES configuration
Joni Baecklund, OH2RBJ for lots of feedback, ideas and documentation
Christian Blattnik, DC0JI for providing a place for DB0CLX
Brigitte Blattnik, DH3MBJ for supporting DJ0ZY, DL6RAI and DC0JI
Robert Chalmas, HB9BZA, for a French version of adv_txt and help files
Diego Serafin, IK3HUK, for the first Italian version of adv_txt and help files
Jonny, DG4MMI, for the German version of adv_txt
Mark Wahl, DL4YBG, for putting special features into TNT
Andrea Fiorentino, I0/N5KME, for the second Italian version of adv_txt
Rich Schmelkin, AE4EJ for running the old CLX mailing list
Ian Maude, G0VGS for helping a lot with the user documentation
Jose Lopez, EA5SW, for a Spanish version of adv_txt
Ignacio Galiana, EA7FPE, for the new version of the Spanish adv_txt
Gerard Parat, F6FGZ for a French version of adv_txt and extensive user manual
Lutz Petschulat, DG0LP, for updating the CLX help files to version 3.02
Heikki Hannikainen, OH7LZB, for running the current CLX mailing list
Luca Palazzo, IW9EXL, for updating the Italian files
Luiz F. Catalan, PP5AQ, for a Portuguese version of adv_txt
Erwin Lemmers, PE1NMB, for a Dutch version of adv_txt and help files
Matthew (Max) George, NG7M, for picking up the FAQ maintenance
Peter Pfann, DL2NBU, for his work on the SHOW/SUN command family
Ulrich Mueller, DK4VW, for his work on the German user manual
Arnold Bosch, PE2AB, for running the CLX web page
Angel Claus, EA7WA, for his help on the Spanish adv_txt
```

Did I forget you? Please let me know! Want to show up in this list too? Ask for the list of jobs to be done! Many details and small jobs could be outsourced if we found the people to do it.

7.5 Frequently Asked Questions

The following is a collection of frequently asked questions by users of the CLX software. The FAQ is now (as of September 1998) being maintained by Matthew (Max) George, NG7M. It will probably be left out from this document in the near future.

Q. What are the timings for CLX setting up connections to other CLX and/or Pavillion-nodes?

A. CLX usually measures the time for how long nothing has been received on a link connection. If this is longer than five minutes, CLX sends a ping (PC51) to the node. Then it waits for another 5 minutes. If nothing is received in return to the ping, it explicitly disconnects the link connection, waits one minute and starts to setup the link again.

Initially, after CLX has been brought up, the software will start making connections after a period of 60 seconds to allow all database activities to finish before the first node-to-node connection is established.

Q. Which services must be started on my system to run CLX?

A. Basically, you need syslogd, portmap and postmaster. postmaster must be running under user postgres, all others as root. Additionally you need one of the two communication packages, either Alan Cox's AX.25 driver or WAMPES. These must be running when CLX is started.

Q. CLX has no monitor. How can I see what my users are doing?

A.

```
$ tail -f ~/logs/io.log
```

if you have configured syslogd correctly. You may also now use the log_monitor as described in section 5.9.3 (Log Monitor).

Q. What Linux versions has CLX been tested on?

A. We have started CLX with Linux 1.1.50 and we are currently using it under 2.0.35.

Q. rpcinfo says:

```
rpcinfo: can't contact portmapper:
RPC: Remote system error - Connection refused
```

What's wrong?

A. `portmap` is not running. Start it from the `rc.*` script when booting Linux.

Q. In the Postgres directory structure, many files have permissions 600 ("`-rw---`"). How can any program touch these files after all?

A. The Postgres database system does all its job via the postmaster process who is handling the communication with any external processes and programs. This is needed for serialisation so that every transaction is finished before the next one starts. Due to this, only the postgres user must have access to the database files. Any user permits etc. are treated internally by the Postgres software.

Q. How come the `kissinit` command works without specifying a speed?

A. 9600 is the default. If you are using another speed on your line, you must set this explicitly with the stty command.

Q. When starting up CLX everything comes up until `con_ctl`, which hangs forever. What's wrong?

A. The two callsigns, the one you specify with kissattch and the one which is encrypted in `~/config/clx_par` must be identical. Otherwise `con_ctl` will hang.

Q. `SH/DX 20` no longer works, but `SH/DX` is still OK. What's happened?

A. Most probably your DX database table has gone wild. This is a bug in Postgres which has not yet been cleared. We have observed a destroyed DX table many times at DB0CLX. Your only chance to correct the problem is destroying and building the table from scratch. Here is how:

```
$ cd ~/db
$ psql clx_db
clx_db=> drop table dx_data;
clx_db=> \i dx_data.cl
clx_db=> \q
$
```

Q. CLX doesn't seem to delete the nodes (in `sh/conf`) after a disconnect. Is it because I'm using an external script? If not can I do something for this?

A. The Nodes listing is a programming problem. I will try to explain:

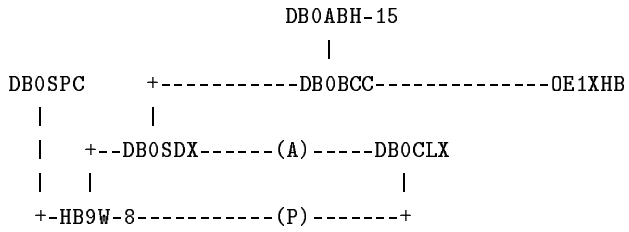
CLX allows loops and multiple connects within a PacketCluster/CLX network. If a node connection is lost but another one still exists into the network, which of the nodes in the list are then no longer "members of the network"? We have tried to solve this question several times without success. So I think CLX currently just follows the node connect/node disconnect (`PC19/PC21`) spots and as long as it still has a valid connection, displays all nodes in the network.

If you can provide us with a good solution to the problem we are ready to implement it.

Q. CLX does not report its own node connection list to other Pavillion PacketCluster nodes? This hides the network behind CLX to their users. Why don't you report this information (through PC38/PC39, PC19/PC21)?

A. The problem you describe (PC38) has in fact nothing to do with PC38. It is only a compatibility issue and for that reason, CLX is not sending any such information to a PacketCluster node. Here is why:

About 9 months ago CLX would send a list of connected nodes out to every active link. Due to CLX's special ability to make multiple connects into a network, some distant nodes already showed up in CLX's Node list. We had a configuration like this:



We found out that whenever DB0CLX reported that it had connected HB9W-8, DB0SDX would no longer try to establish its own connection to HB9W-8. Obviously, the PacketCluster software anticipated that the connection already existed.

We had this same situation between DB0BCC, DB0SDX and DB0CLX so it is certainly a bug/problem in the Pavillion software (which does not know about multiple links).

For this reason we decided that CLX would not report any network information to neighbouring Pavillion nodes. However, it does fully report to adjacent CLX nodes.

Q. Is there a CLX mailing list?

A. Yes! Thanks to Heikki Hannikainen, OH7LZB, CLX has its own mailing list. See 3.1 (CLX Mailing List) for details on how to subscribe.

By the way, the current list of CLX users can be found elsewhere in this document in 7.3 (CLX User List).

Q. How can I restart CLX when a process has died? `clx -s` followed by `clx -u` doesn't work.

A. The reason for this is that (a) one or more of the processes hung and did not clear its rpc port address, or (b) some of the shared memory areas is not free. What can you do?

There is now a `-x` to the CLX script which allows clearing resources. Clearing rpc ports, however, requires root privileges.

```
$ clx -x
You must be root to clear rpc ports.
Password:
```

Now you should be able to restart CLX without rebooting the machine.

Q. I have only a DOS machine on the Internet. How can I get the big CLX archive to my Linux box?

A. There are several ways. One that I have used often in the past is by using `PKZIP` on the DOS and `unzip` on the Linux side.

Prerequisites: - PKZIP version 2.04 on DOS machine - unzip version 5.12 on Linux machine.

1. Download the file on your DOS machine
2. Copy the file on two disks.

```
C:\TMP>dir *.tgz
CLX_301  TGZ     2,564,043 07-20-97   9:06p

C:\TMP>pkzip -& a:clx.zip clx_301.tgz

Creating ZIP: A:CLX.ZIP
  Adding: CLX_301.TGZ  Deflating ( 2%)
Insert disk #2 - Press a key when ready
, done.

C:\TMP>
```

3. Go to your Linux machine and copy the two disks, rename the file from disk 1 to `clx1.zip` and the file from disk 2 to `clx2.zip`:

```
$ mcopy a:clx.zip clx1.zip
Copying clx.zip
$ mcopy a:clx.zip clx2.zip
Copying clx.zip
```

4. Concatenate the two files into one big `clx.zip` file:

```
$ cat clx1.zip clx2.zip > clx.zip
```

5. Now unpack the clx.zip file, ignore the warnings.

```
$ unzip clx.zip
Archive:  clx.zip
warning [clx.zip]:
  zipfile claims to be last disk of a multi-part archive;

  attempting to process anyway, assuming all parts have been
  concatenated together in order.  Expect "errors" and warnings...
  true multi-part support
  doesn't exist yet (coming soon).
warning [clx.zip]:  extra 1457664 bytes at beginning or within zipfile
  (attempting to process anyway)
file #0:  bad zipfile offset (local header sig):  1457668
  (attempting to re-compensate)
  inflating: CLX_301.TGZ
$ ll CLX_301.TGZ
-rw-r--r--   1 ben      users     2564043 Jul 20 21:06 CLX_301.TGZ
```

You should now have clx_301.tgz on your Linux box.

Q. What do the occasional messages like

```
icl_com/us_ctrl.cc: Message is not correct "PC50"/
```

in my io.log mean?

A. PacketCluster knows a special type of external connections using the `-EXT` switch in the node definition. Due to a bug in the AK1A software, the PC50 will be sent in a different format which is not understood by CLX.

Q. My system is crashing very often for unknown reasons. Other CLX sysops report that the version I am using is generally stable. What can I do?

A. We have found that once in a while, the Postgres database can become slightly corrupted - probably due to previous crashes or other reasons. That means, the data is available and all looks OK but some datum deep in the database is broken and when CLX reads it (due to some innocent user command), a string variable may be exhausted, a float may be returned where an integer was expected etc. etc. As CLX is never checking the input data received from the database, a program may crash from this bad data.

What has cured the problem in the past several times was backing up the database, destroying it completely, recreating it and re-reading the data. How do you do it? First you need some time (maybe several hours, depending on your data) and you need to shutdown CLX. Having done that, issue the following commands (all as user `clx_us`):

```
$ bup_db -s
$ clx_db
$ bup_db -r
$ clx_idx
```

While backing up the files is normally pretty fast, re-reading the data may take a while.

This treatment was successful for DB0BCC when CLX version 4.00b turned out to be pretty stable elsewehere but not at DB0BCC. After re-reading the data, the system would stay up for up to 16 days.

Q. How can I turn off the screen blanker?

A. Use `/bin/setterm -blank 0` in your Linux startup scripts.

Q. What does the message `"kernel: Unable to load interpreter"` mean? I find this in my log when CLX crashed.

A. This is a very definitive message from the operating system kernel saying that it cannot execute a new program because it cannot load the ELF interpreter. We have observed that with version 4.00 of CLX where a bug in one of the programs ate up more and more file descriptors (FDs) and in the end (after a few hours, sometimes days), there were no more FDs available. Then all kinds of strange things would happen. Fortunately this bug was found very quickly but when such a message ocurs again, you should check for the number of FDs currently used with the following command:

```
# echo /proc/[0-9]*/fd/* | wc -w
```

This command must be executed as root. When freshly started, CLX needs about 300 FDs. The limit in the unmodified Linux kernel is 1024.

7.6 Known Bugs in the CLX software

Here is a list of known bugs in the current CLX software. If you happen to find a bug not listed here, please let us know.

- Node callsigns do not disappear from the `show/configuration` display even after they were disconnected. This is a problem which needs a concept. Due to CLX's ability to support multiple links, it is unclear when a node is no longer available if one link fails.
- When using `REPLY` to answer a mail message you received from a station at a different node, your return message will not be forwarded.
- Telnet link connections have an echo which produce problems within CLX.

7.7 Bugs and Bug Reporting

Nobody is perfect. So is this software. We have put a lot of work and energy into this project but yet some problems are left to be unleashed. CLX has been on the air at DB0CLX for more than four years now. In the mean time, many bugs were fixed, new ones introduced and fixed again. At this time we have reached a point where we believe there are no major bugs left (but you may prove us wrong).

If you find bugs, please try to describe them in a detailed way and let us know. Please be as specific as possible and include screen shots, logs etc. with your report. Send the report by email to *clx@dl6rai.muc.de*.

Sometimes one of the CLX processes dies silently. You will see that by inspecting `io.log` or `error.log` and seeing a message like this:

```
snd_ctl/rpc_send.cc -> con_ctl, 0 - 1/dj0zy: Connection refused
```

This usually indicates that `con_ctl` has died and `snd_ctl` is not able to deliver a telegram to this process.

In such a case it is very valuable for us to have a copy of the last 10-20 messages in the `io.log`. Please send this together with your error report.

This software is developed by DJ0ZY and DL6RAI. DJ0ZY is doing the C++ and Postgres core programming. So his activities are mainly in the `~/bin` and `~/db` directories. DL6RAI's job is maintaining the Perl scripts in `~/bin`, `~/tools`, `~/exec` and the documentation in `~/doc`. This information should help you to direct your bug reports either to Franta or to me. If in doubt, send your message to *clx@dl6rai.muc.de*.

7.8 Wish List

A list of features currently missing in CLX which are going to be implemented next.

- Making clx_watchdog send a warning message to users, one minute before it's shutting down CLX.
- An extended DX spot line which gives country names, beam heading etc.
- A SHOW/MUF or SHOW/PROPAGATION command (OE1TKW is working on it).

Index