VMS XNS Series 2000/3000 Software Guide

Description

This package contains the documentation needed to install and use XNS software on a VAX running the VMS operating system. It is comprised of:

VMS XNS Software Notes, part number 5001-002-001-0

Although VMS XNS software release contains all the capabilities of the standard workstation release, some features have not been fully implemented, it requires certain workarounds. These notes document workarounds associated with the release and minor bugs not mentioned in the list of known problems in the documentation.

VMS XNS Software Installation Series 2000, part number 5001-004-001-0

This document discusses the XNS software Revision N2.3 for VAX/VMS. It contains software installation and testing procedures, file descriptions, and diagnostic procedures.

VMS XNS User's Guide, part number 5001-003-001-0

This document describes XNS software library and utilities programs for VMS. It discusses procedural interface, usage, problems, and restrictions and contains descriptions of the XNS library routines and the three client application programs: *xx, xlogin, and xcp*.

Silicon Graphics, Inc.
2011 Stierlin Road
Mountain View, CA 94043

Document Number 007-0302-010

VMS XNS Software Notes
August 1985

Although this release contains all the capabilities of the standard workstation release, some features have not yet been fully implemented (these features will be implemented soon); it requires the workarounds mentioned below. This page also includes a discussion of some minor bugs not mentioned in the list of known problems in the documentation.

Utilities

Running on VMS Communicating with a workstation

xx

Since XNSEOF transmits 2 garbage characters in addition to the EOF, (see the description of xnseof in the following Libraries section) the following problem occurs:

```
$ xx sgi_workstation cat > /tmp/foo
hello sailor
^Z
$
```

If you check the file /tmp/foo on sgi_workstation, it has 2 extra characters at the end.

xcp

A lost connection error will occur if the following conditions are ALL met:

1. Transferring multiple files from VMS to the workstation (via wildcards or using a list of input files on the command line)
2. At least one of the files to be transferred has the following format:
    A. Fixed length records
    B. Truncated last record

The suggested workaround is to transfer the file(s) with the fixed length records with individual xcp commands.

Running on a Workstation Communicating with VMS

xlogin

If you `xlogin` to VMS, you will not return control to the workstation when logging out. To do this, you must enter " ." as the first two characters on a line (they will not echo). You may do this either before or after entering "logout".

If you `xlogin` to your VMS host from a workstation, then `xlogin` back to your workstation, either the connection will fail or you will be logged off VMS. You will need to use " ." as described in the previous paragraph to return control to your workstation.

xx

If you `xlogin` to your VMS host from a workstation, then use `xx` to login back to your workstation, either the connection will fail or you will be logged off VMS. You will need to use " ." to return control on your workstation (see xlogin above).

Xx is not available for talking to VMS from the workstation yet. If you need information from your VMS host, you must use xlogin for now.

xcp

xcp cannot be used directly from the workstation. Instead, *xlogin* to your VMS host, and run `xcp` from there, `xcp` on VMS will transfer tiles to/from your workstation.

Libraries

xnslib.olb modules

xnseof

XNSEOF transmits 2 garbage characters in addition to the EOF.

xclose

If a workstation sends information (using xwrite or xnswrite) then closes the connection, the VMS host reading (using xread or xnsread) from the connection will not receive all the data that was sent, and *may* cause an error.

VMS XNS Software Installation Series 2000

Release N2.3

Silicon Graphics, Inc.
630 Clyde Court
Mountain View, CA 94043

Document Number 5001-004-001-0

Technical Publications:

Marcia Allen
Robin E. Florentine
Steven A. Locke
Susan Luttner
Celia Szente
Diane M. Wilford
Glen Williams

VMS XNS Software Installation Series 2000
Document number: 5001-004-001-0

CONTENTS

1. Introduction

This document discusses the XNS software Revision N2.3 for VAX/VMS. Read this document thoroughly before proceeding.

There are three chapters and two appendices in this document.

- Chapter 1 contains this introduction.
- Chapter 2 covers reading the tape.
- Chapter 3 contains information on testing the software.
- Appendix A contains the file descriptions.
- Appendix B covers diagnostics.

Silicon Graphics provides a comprehensive product support and maintenance program for IRIS Series 2000 Products. For further information, contact Customer Service through the Geometry Hotline.

Silicon Graphics Geometry Hotline	
(800) 252-0222	U.S. except California (toll-free)
(800) 345-0222	California (toll-free)
(415) 962-0606	Worldwide (collect)

2. VMS XNS Installation

2.1 Reading the Tape

The installation tape is a standard VAX/VMS backup tape, written at a density of 1600 BPI. Be sure to remove the write ring before mounting the tape, if one is present. If you also ordered the FORTRAN remote graphics library, install it AFTER installing this package. When this package is successfully installed, you can delete any previous versions of this software.

1. Login as the system manager, and type the following:

```
$ ASSIGN _MTA0: XNSTAPE ! Substitute your tape drive for _MTA0:
$ MOUNT/FOREIGN/NOWRITE/DENSITY=1600 XNSTAPE
 ! For binaries only
$ BACKUP/VERIFY/LOG XNSTAPE:XBN23.BCK SYS$SYSDEVICE:[IRIS.XNSN23]
 ! For binaries and sources
$ BACKUP/VERIFY/LOG XNSTAPE:XSN23.BCK SYS$SYSDEVICE:[IRIS.XNSN23]
$ !Be sure to use SYS$SYSDEVICE.
```

2. If there are any problems reading in the distribution tape:
   - Check the density settings on the drive.
   - Clean the tape heads.
   - Try another drive, if you have one.
   - Call your Silicon Graphics representative for a replacement tape.

2.2 Installing the Software

CAUTION:
Since it is possible the system may crash when installing the software, you should install the software when it is safe to crash the system.

1. Use a hard-copy terminal tor your installation.
2. Set your default to the device and directory containing the XNS software:

```
$ SET DEFAULT SYS$SYSDEVICE:[IRIS.XNSN23]
```

3. Silicon Graphics provides a command file that automatically updates the software. To execute this file, type:

```
$ @XNSINSTAL
```

3. Verification

This chapter provides procedures tor verifying that the XNS communications software functions properly.

3.1 Terminal

1. Go to an IRIS terminal that has the Ethernet cable attached.

   Push the button labelled RESET. The following prompt will appear on the top left part of the monitor screen:

   ```
   iris >
   ```

   If you do not get the prompt, you may have a hardware problem. Check the following:

   a. Configuration switches (see *IRIS Terminal Guide, Series 2000* and appropriate Software Distribution documents).

   b. If you're upgrading your IRIS, make sure the upgrade was completed (that is, the Ethernet tap is secure, etc.).

2. To download the IRIS terminal with the graphics monitor, enter the following:

   `iris> n` *yourhostname:filename*

   In the above command, *yourhostname* is the exact name you've given your host, and *filename* is GL2T22. This file is included as part of the Remote FORTRAN Software Distribution.

   When downloading the GL2T22 file, the following message appears:

   ```
   Downloading.....................:
   ```

   Next, the screen clears, and the following prompt appears:

   ```
   Network iris terminal GL2 ...
   Connect to what host?
   ```

   If you make any typing errors, use backspace, not erase, to delete characters.

Downloading Problems

- If the "Downloading ..." message does not appear:
    a. Check that the Unibus adapter, base CSR, and interrupt vector addresses on the Interlan board agree with the numbers in the XNSLOAD.COM file.
    b. Make sure that the NPR jumper has been removed from the Unibus. (CA1 to CB1 on the backplane).
    c. See Appendix B, Terminal Diagnostics.
- If the "Downloading ..." message appears, but the colon doesn't appear:
    a. Push the RESET button, and try again.
    b. Check the SYSGEN parameter BUFIO, and make sure the value is greater than 512 (decimal).
- If the error message "no boot server responding" appears:
    a. Double check that you spelled the hostname exactly as found in the XNSLOAD.COM file.
    b. See Appendix B, Terminal Diagnostics.
- If the error message "open file connection error" appears:
    a. Double check your spelling of **GL2T22** or **dbiris**.
    b. See Appendix B, Terminal Diagnostics.

3. The following prompt appears:

```
Network iris terminal GL2 ...
Connect to what host?
```

Enter:

yourhostname

The screen clears, and you will have the standard login from VMS. You should now be successfully logged on. (The delete character is now your standard delete character.)

The terminal is set up to emulate a VT52; you should enter:

```
$ SET TERMINAL/DEVICE_TYPE=VT52
```

> **NOTE:** The emulation has a few characteristics above and beyond the "usual" VT52, see the *IRIS Terminal Guide, Series 2000* for more information.

If you incorrectly enter your password, or somehow manage to get the "User authorization failure" message, you will have to reboot the terminal with the RESET button.

4. If you have the Silicon Graphics Remote FORTRAN Software Distribution, set your default to that of the remote demo programs, and try running the remote demos (SQIRAL, STAR, and FLOATS). See the FORTRAN distribution file FRTINSTAL.TXT on how to run these programs.

3.2 Workstation

To test the software on the IRIS Workstation, try some of the utilities, such as **xcp** and **xlogin**. You must use level GL2-W2.3 or greater software.

1. On the IRIS workstation, enter the following:

```
xx host command    !eg: xx olympus date
xcp file host:filename
xcp host:file filename
xlogin host
```

2. Repeat the above steps on the "host" communicating to the IRIS workstation. If the files are copied correctly, XNS is working.

Appendix A: File Descriptions

These files are contained on the host machine (e.g., VAX) software distribution tape.

`BRCV.EXE`	Tests whether any broadcast messages are being received from the IRIS. This image will not boot the IRIS terminal.
`BROADECHO.EXE`	Tests whether the broadcast message that it sends out to the net makes it back in.
`ECHO.EXE`	Tests to see if the echo packet it sends out addressed to itself makes it back. (Loop around test).
`SETHOST.EXE`	Process that starts up the **xns server** processes for the IRIS terminal, and sets the **hostname**.
`SGBOOT.EXE`	Creates a detached process that boots IRIS terminals.
`SGBOUNCE.EXE`	Creates a detached process that handles bounce packets.
`SGEXEC.EXE`	Creates a detached process that handles connections on the EXEC socket.
`STARTXNS.COM`	Command file to (re)start the XNS server processes.
`STOPXNS.COM`	Command file to stop the XNS server processes.
`XCP.EXE`	File transfer utility.
`XNSD.EXE`	XNS detached process.
`XNSDRIVER.EXE`	XNS driver that does the I/O to the Interlan board.
`XNSINSTAL.COM`	Command file for installing the XNS software. See Section 2 in this manual.
`XNSINSTAL.TXT`	This file.
`XTDRIVER.EXE`	The "terminal" driver associated with the XNS driver that queues requests to the XNS driver.

Appendix B: Terminal Diagnostics

Run this software only if you install the XNS software and determine that you cannot boot an IRIS terminal. These diagnostics do not apply to the IRIS workstation.

If ail these diagnostics work, the IRIS is able to transmit broadcast packets, and the VAX can transmit and receive packets. The diagnostics do not determine whether the IRIS can receive packets.

First remove the bootserver processes that are running on the system, by entering the following:

```
$ SET DEFAULT XNS$DIR
$ @STOPXNS
```

You are now ready to run the diagnostics. Run the diagnostics in the following order: BROADECHO, ECHO, and BRCV.

To restart the bootserver processes, enter:

```
$ SET DEFAULT XNS$DIR
$ @STARTXNS
```

B.1 BROADECHO

This program writes a broadcast packet out the net, and then attempts to read that packet back in. To run the diagnostic, enter:

```
$ RUN BROADECHO
```

The following output is what shows up on the screen if **broadecho** has run successfully:

```
This program is a loop-around test to find out if we can send bounce
packets out to the Ethernet and retrieve them. Only broadcast
is used.

ALL NUMBERS WILL BE REPORTED IN HEX

Test pattern is abcdefghijklmnopqrstuvwxyz0123456789
Opening channel to XNS
Channel successfully opened, using 50
Setting physical address on net
VMS net address is  2   7   1   0   7   9d

Establishing self as local bounce server


Now the official bounce server

Now we're about to send out a bounce packet

Immediate destination:

ff  ff  ff  ff  ff  ff

Immediate source:

2  7  1  0  7  9d

Internet packet type should be  80  16

Internet packet type:

80 16


Data in packet:
```

```
a b c d e f g h i j k l m n o p q r s t u v w x y z 0 1 2 3 4 5 6 7 8 9

Posting a read for a bounce packet

We got a packet!

The following is a bounce packet description:

The number of characters read is 80

Immediate destination:

ff  ff  ff  ff  ff  ff

Immediate source:

2  7  1  0  7  9d

Internet packet type should be  16  80

Internet packet type:

16  80

Data in packet:

  a b c d e f g h i j k l m n o p q r s t u v w x y z 0 1 2 3 4
5 6 7 8 9
```

Error Descriptions

```
Establishing self as local bounce server
Now the official bounce server
```

If the above messages do not appear, but instead "Can't become the bounce server..." appears, stop the **xns server** processes. To do this, enter the following commands:

```
$ SET DEFAULT XNS$DIR
$ @STOPXNS
```

If the line "Posting a read for a bounce packet" appears, but the line "We got a packet!" doesn't appear, the VAX Ethernet board is either not broadcasting or receiving broadcast packets correctly.

B.2 ECHO

This program writes a packet addressed to itself out to the net, then attempts to read it back in. To run the diagnostic, enter the following:

```
$ RUN ECHO
```

The following output is what shows up on the screen if echo has run successfully:

```
This program attempts to find out if we can send bounce packets out to the
Ethernet and grab them back again - four basic loop around test, broadcast
is not used - only the host machine is addressed


ALL NUMBERS WILL BE REPORTED IN HEX

Test pattern is abcdefghijklmnopqrstuvwxyz0123456789

Opening channel to XNS

Channel successfully opened, using 50

Getting physical address on net

VMS net address is  2   7   1   0   7   9d

Establishing self as local bounce server

Now the official bounce server

Now we're about to send out a bounce packet

Immediate destination:

2  7  1  0  7  9d

Immediate source:

2  7  1  0  7  9d

Internet packet type should be 80  16

Internet packet type :
```

```
80  16

Data in packet:

a b c d e f g h i j k l m n o p q r s t u v w x y z 0 1 2 3 4 5 6 7 8 9


Posting a read for a bounce packet

We got a packet back!

The following is a bounce packet description:

The number of characters read is 80


Immediate destination:

2  7  1  0  7  9d

Immediate source:

2  7  1  0  7  9d

Internet packet type should be   16   80

Internet packet type:

16   80

Data in packet:


 a b c d e f g h i j k l m n o p q r s t u v w x y z 0 1 2 3 4
5 6 7 8 9
```

Error Descriptions

```
Establishing self as local bounce server
Now the official bounce server
```

If the messages above do not appear, and instead the message "Can't become the bounce server ..." appears, you should stop the **xnsserver** processes, by entering the following commands:

```
$ SET DEFAULT XNS$DIR
$ @STOPXNS
```

If the line "Posting a read for a bounce packet" appears, but the line "We got a packet!" doesn't appear, then the VAX Ethernet board is either not sending or receiving packets correctly.

After you are finished running the diagnostics, enter:

```
$ SET DEFAULT XNS$DIR
$ @STARTXNS
$ SHOW SYSTEM
```

The SGBOOT, SGBOUNCE, and SGEXEC processes will return to the display.

B.3 BRCV

This program sets up a "read" on the net, looking tor the broadcast packet that is sent from the IRIS terminal when given the *iris>n host:file* command. The IRIS terminal will not boot up, but BRCV should report that it received a packet. To run the diagnostic, enter:

```
$ RUN BRCV
```

The following output is what shows up on the screen if **brcv** has run successfully:

```
This program checks for bounce packets on the Ethernet
but will not boot anything. It is strictly a test.

ALL NUMBERS WILL BE REPORTED IN HEX

Opening channel to XNS

Channel successfully opened, using 50

Getting physical address on net

VMS net address is  2   7   1   0   7   9d

Establishing self as local bounce server

Nov the official bounce server

Posting a read for a bounce packet

Please go to the IRIS and enter a boot command like:

iris> n aardvark:boink

Bounce packet received!

The following is the bounce packet description:

The number of characters read is 74

Immediate destination:

ff  ff  ff  ff  ff  ff
```

```
Immediate source:

8   0   14  0   20  38

Internet packet type:

16  80

Data in packet:

C : a a r d v a r k : b o i n k
```

Error Descriptions

Establishing self as local bounce server No* the official bounce server

If the above messages do not appear, and instead the message "Can't become the bounce server ...", appears, you should stop the xnsserver processes, by entering the following commands:

```
$ SET DEFAULT XNS$DIR
$ @STOPXNS
```

If the diagnostics ECHO and BROADECHO run successfully, but the message "Bounce packet received!" doesn't appear after entering the boot command on the IRIS terminal, then the IRIS terminal Ethernet board is not working properly. It is either not broadcasting correctly, or is not responding to it's Ethernet address (found on the Ethernet board).

VMS XNS User's Guide

Version 1.0

Silicon Graphics, Inc.
2011 Stierlin Rd.
Mountain View, CA 94043

Document Number 5001-003-001-0

Technical Publications:

Marcia Allen
Robin E. Florentine
Steven A. Locke
Susan Luttner
Celia Szente
Diane M. Wilford
Glen Williams

VMS XNS User's Guide
Document number: 5001-003-001-0

CONTENTS

1. Introduction

Silicon Graphics Ethernet software for the VMS operating system consists of a pair of device drivers, several application programs, and a software library. The XNS device driver is written for an Interlan 1010 device. It provides a VMS interface to the device and an implementation of the Xerox Sequenced Packet Protocol (SPP). The XT device driver is used in conjunction with the XNS device driver to provide remote terminal service on VMS. The XT driver acts as a bridge between the SPP transport protocol and the VMS terminal class driver.

Client application programs are provided to run commands on other computer systems, transfer files between systems, and log onto other systems over the Ethernet. These programs are `xx`, `xlogin`, and `xcp`; they are described in Appendix B. Note that these programs must be defined as foreign commands. Server application programs are provided on VMS that allow users on other computer systems to run commands on VMS, transfer files, and to log into VMS. In addition, server programs are provided to support the downloading of Silicon Graphics diskless systems.

The software library contains C routines that provide access to the SPP functions of the XNS driver. These functions include creating SPP connections to other hosts, and listening for incoming connections as well as the expected read-write I/O functions.

This document describes XNS software library and utilities programs for VMS.

There are three chapters and two appendices in this document.

- Chapter 1 contains this introduction.
- Chapter 2 contains information on procedural interface.
- Chapter 3 contains information on usage.
- Appendix A describes known problems and restrictions.
- Appendix B contains descriptions of the XNS library routines and the three client application programs xx, xlogin and xcp.

Silicon Graphics provides a comprehensive product support and maintenance program for Silicon Graphics Products. For further information, contact Customer Service through the Geometry Hotline.

Silicon Graphics Geometry Hotline	
(800) 252-0222	U.S. except California (toll-free)
(800) 345-0222	California (toll-free)
(415) 962-0606	Worldwide (collect)

2. Procedural Interface

2.1 Introduction

The XNS library contains Ethernet driver access routines plus other functions that are used in network utilities provided by Silicon Graphics. These routines are documented by manual pages in Appendix B of this manual. The XNS library routines access the device driver using `sys$qio` calls. If you need to write your own access routines, refer to Section 3.3.

2.2 XNS Subroutine Library

All the routines are found in `xnslib`, described in Appendix B.

Two external variable declarations are needed in the application program when using any of the following routines: `gethostname`, `xclose`, `xnsconnect`, `xnseof`, `xnsfile`, `xnslisten`, `xnsread`, `xnswrite`, `xread`, or `xwrite`. They should look like this in the user program:

```
int vmserrno, xns_efn;
main(argc, argv)
{
...
}
```

`Vmserrno` is the last status code passed back by a VMS system call in the XNS library. You can show the error returned with the SYS$PUTMSG call, or by way of DCL with the line,

```
exit(vmserrno);
```

`Xns_efn` is the event flag number you want the XNS routines to use. All the XNS calls are synchronous, so you can set `xns_efn` to 0.

2.3 The QIO Interface

This section describes the VMS C-language QIO interface to the SGI XNS driver.

The following list contains the VAX11 C include files needed to define structures and system constants:

```
descrip.h
iodef.h
psldef.h
signal.h
ssdef.h
stsdef.h
```

Function codes for the P1 parameter in IO$_ACCESS follow:

```
#define XNS_SET_HOST                1/* Set Host Name*/
#define XNS_LISTEN                  2/* Listen for connection*/
#define XNS_TRACE_INFO              3/* Get trace info*/
#define XNS_BOUNCE_SERVER           4/* Register as the bounce server*/
#define XNS_GET_PHYSADDR            5/* Get physical Ethernet address*/
#define XNS_FAST_IO                 6/* Set fast I/O mode*/
#define XNS_CONNECT                 7/* Connect to foreign host*/
#define XNS_GET_HOST                8/* Put Host Name into str descr*/
```

The following are offsets into the VMS IOSTAT block (see below for the VMS IOSTAT block):

```
#define IOSTAT          0
#define BYTECOUNT       1
#define DTYPE           2
```

Variables expected by some of the code below follow:

```
int vmserrno;                       /* VMS System service error return */
int xns_efn;                        /* io event flag used by vms*/
unsigned short iosb[4];             /* vms iostat (status block) */
unsigned short channel;             /* vms io channel */
char buffer[];                      /* user io buffer */
unsigned int chan;                  /* ins channel */
int size;                           /* io byte count */
```

For each of the calls below, success is indicated if `(vmserrno&STS$M_SUCCESS)` is set. An error is indicated if the `STS$M_SUCCESS` bit is clear. In addition, `iosb[IOSTAT]&STS$M_SUCCESS` should be checked as noted below.

Assigning an IO Channel

Use the following call to obtain a VMS IO channel for the XNS driver:

```
static $DESCRIPTOR(xns0_chan_desc, "XNS0:");
vmserrno = sys$assign(&xns0.chan_desc,*channel,PSL$C_USER,0);
iosb[IOSTAT] & STS$M_SUCCESS
```

Making an XNS Connection

Connect to the host addressed by the `xns_setup` descriptor. `Xns_setup` is described on the manual page for `xnsphysconnect` in Appendix B.

```
struct { int size; struct xns_setup *ptr; } descr;
descr_size = sizeof(descr);
vmserrno = sys$qiow(xns_efn,chan,IO$_ACCESS,iosb,0,0,
        XNS_CONNECT,&descr,0.0.0.0);
```

Listening for an XNS Connection

`Chan` should be an active IO channel for the XNS driver. Success is returned when the driver accepts an incoming connection request for the specified socket.

```
vmserrno = sys$qio*(xns_efn,chan,IO$_ACCESS,iosb,0,0,
        XNS_LISTEN,socket,0,0,0,0);
```

Netread

Read SIZE bytes from the network into BUFFER.

The current XNS datastream type is returned in `iosb[DTYPE]`. The byte count read from the driver is returned in `iosb[BYTECOUNT]`. The code below returns 0 on end of file, -1 on failure, and returns the transfer count on success.

```
vmserrno = sys$qiow(xns_efn,chan,IO$_READVBLK,iosb,0,0,
           buffer,size,0,0,0,0);
if (!(iosb[IOSTAT] & STS$M_SUCCESS)) {
   if (iosb[IOSTAT] == SS$_ENDOFFILE)
      return(0);
   vmserrno = iosb[IOSTAT];
   return(-1);
}
return(iosb[BYTECOUNT]);
```

Netwrite

Write SIZE bytes from IO BUFFER using datastream type DTYPE. The code below returns -1 on failure and returns the byte count on success.

```
vmserrno = sys$qiow(xns_efn,chan,IO$_WRITEVBLK,iosb,0,0,
           buffer,size,dtype,0,0,0);
if (!(vmserrno* STS$M_SUCCESS))
   return(-1);
if (!(iosb[IOSTAT] & STS$M_SUCCESS)) {
   vmserrno = iosb[IOSTAT];
   return(-1);
}
return(size);
```

Terminate Connection

Connections are terminated when a program exits or when it explicitly deassigns the VMS channel.

```
vmserrno = sys$dassgn(chan);
```

3. Examples

3.1 Introduction

This section contains examples of using the XNS procedural interface in a program that tests the flow control software for XNS. If the program is given an argument, it becomes the sender of data. With no argument, it is the receiver. The test should be run twice, once for each direction of data flow.

The routine `xnsconnect` is used by the sender to connect to a socket. The routine `xnslisten` is used by the receiver to connect to a socket. The routines `xread` and `xwrite` are used to read and write data, `xclose` is used to close the socket connection. Note that under SYSTEM V, `xread`, `xwrite`, and `xclose` do not exist and read, write, and close are used instead.

USAGE:

flow [**-d**] hostname

SWITCHES:

-d turns on debugging mode, prints out packets as they are sent or received.

NOTES:

To compile properly the following variables must be defined:

```
IRIS 2400 :  MC68000 SYSTEMS
VAX UNIX  :  VAX UNIX4_2
VAX VMS   :  vms
```

When running UNIX, environment variables MACHINE and SYSTEM are set to the appropriate values and the Makefile uses them. When running VMS, the compiler defines VMS, so it works by itself.

```
/*********************************************************************
/* compile time defines
/*********************************************************************/
#include <stdio.h>
#include <errno.h>

#ifdef vms
#define errno vmserrno
#else
#define xread read
#define xwrite write
#define xclose close
#endif

#ifdef MC68000
#define byte_swap_long(i)
#else
#define byte_swap_long(i) {
   register char c,*cp;
   cp = (char *)&i;
   c = cp[0];  cp[0]=cp[3];   cp[3]=c;
   c = cp[1];  cp[1]=cp[2];   cp[2]=c:
}
#endif

#define MYSOCK 223

/*********************************************************************
/* global variables
/*********************************************************************/

short debug = 0;                        /* debug flag*/
short pause_flag = 0;                   /* pause flag*/
int COUNT = 120;                        /* number of packets to test*/
int error_count = 0;                    /* cumulative error count*/
extern int errno;                       /* external error number*/
```

```
/*********************************************************************
/*       MAIN program
/*********************************************************************/

main (argc,argv)
   int argc;
   char **argv;
{
   register int i,j,f,sender;

   /*********************************************************************
   /* parse command line
   /*********************************************************************/
   while (--argc >0 && **++argv == '-') {
         register char *token;

         for (token = argv[0] + 1; *token; token++)
         switch (*token) {
            case 'd':
                              debug = 1;
                              break;
            default:
                              fprintf (stderr, "illegal option %c\n", *token);
                              break;
         }
   }
   if (argc > 1) {
         fprintf (stderr, "Usage: flow [-d] [hostname]\n");
         exit (0);
   }
```

```
/*********************************************************************
/* establish network: connect or listen
/*********************************************************************/

if (sender = (argc > 0)) {
         begin_notice ("Performing ins connect operation");
         f = xnsconnect(*argv,MYSOCK);
         fprintf (stderr, "(%d) ",f);
         if (f >= 0) end_notice ();
         else {
            fail_notice ();
            goto end_of_test;
         }
}
else {
         begin_notice ("Performing ins listen operation");
         f = xnslisten(MYSOCK);
         fprintf (stderr,"(%d) ",f);
         if (f >= 0) end_notice ();
         else {
            fail_notice ();
            goto end_of_test;
         }
}

for (i=0; i<4; i++) {
         pause_flag = i & (sender ? 1:2);
         fprintf (stderr, "Performing flow control test #%d (%s)... ",
                                  i,pause_flag ? "PAUSING ON" : "pausing off");
         fflush (stderr);
         sender ? bench_write(f) : bench_read(f);
         end_notice ();
}

/*********************************************************************
/* close up the connection
/*********************************************************************/

begin_notice ("Performing close operation");
f = xclose(f);
fprintf (stderr, "(%d) ",f);
if (f >= 0) end_notice ();
else fail_notice ();
```

```
  /*****************************************************************
   /* count the number of errors and exit appropriately
   /*****************************************************************/

end_of_test:
   if (error_count > 0) {
         fprintf (stderr, "Test FAILED, %d errors encountered.\n",error_count);
         exit (1);
   }
   fprintf (stderr,"Test passed.\n");
}

bench_write (f)
   register int f;
{
   register long i,err;
   long j;

   for (i=0: i<COUNT; i++) {
         j = i;
         byte_swap_long(j);
         if ((err=xwrite(f,&j,4)) != 4) {
            fprintf (stderr,"\nERROR: xns write failed (%d,%d)\n", err,errno);
            error_count++;
            return;
         }
         if (debug) {
            fprintf (stderr,"%d ",i);
            fflush (stderr);
         }
         pause (i);
   }

bench_read (f)
   register int f;
{
   register long i,err;
   long j;

   for (i=0; i<COUNT; i++) {
         if ((err=xread(f,&j,4)) != 4) {
            fprintf (stderr, "\nERROR: xns read failed (%d,%d) \n" , err,errno);
            error_count++;
            return;
         }
```

```
         byte_swap_long(j);
         if (debug) {
            fprintf (stderr,"%d ",i);
            fflush (stderr);
         }
         if (i != j) {                    /* compare data with expected result*/
            fprintf(stderr, "\nERROR: xns data error (read %d should be %d)\n",
                                          j,i);
            error_count++;
            return;
         }
         pause (i+3);
   }
}

pause (j)
   register int j;
{
   register int i;

   if (!pause_flag) return;
   switch (j&7) {                         /* wait for a pseudo-random while*/
         case 1:
            sleep (1);
            break;
         case 2:
            sleep (2);
            break;
         case 4:
            for (i=0;i<5000;i"+);
            break;
         case 5:
            for (i=0;i<50000;i++);
            break;
         default:
            break;
   }
}

begin_notice (t)
   char *t;
{
   continue_notice (t);
   continue_notice (" ... ")s;
}
```

```
continue_notice (t)
   char *t;
{
   fprintf (stderr,t);
   fflush (stderr);
}

end_notice ()
{
   continue_notice ("done.\n");
}

fail_notice ()
{
   continue_notice ("FAILED.\n");
   error_count++;
}
```

Appendix A: Known Problems and Restrictions

If you have trouble downloading the software on your IRIS terminal, press the RESET button.

All I/O requests are being tunneled through the terminal driver. This means that if you are doing heavy I/O (especially with the graphics library) the system response could become sluggish to the rest of the users. The remote graphics library uses buffers which are almost always within the limits, but you may want to monitor the BUFIO limit. To speed up the system response, use the `setfas` subroutine.

This software release requires that the R1C or R1B prom sets are installed on the IRIS GL1 terminal.

CAUTION
The driver is not reloadable. If you attempt to load the driver again after it has been loaded, it may crash the system.

Appendix B: Manual Pages

NAME

bcmp – bit and byte string operations

SYNOPSIS

bcmp(b1, b2, length)
char *b1, *b2;
int length;

DESCRIPTION

Bcmp compares byte string *b1* against byte string *b2*, returning zero it they are identical, non-zero otherwise. Both strings are assumed to be *length* bytes long.

BUGS

The *bcmp* routine takes parameters backwards from *strcmp*.

NAME

bcopy – copies 'byte_count' bytes from 'source' to 'target'.

SYNOPSIS

void bcopy(source, target, byte_count)
char *source, *target;
int byte_count

DESCRIPTION

Return value: has no return value.

NAME

bytesize – counts the number of bytes in a null terminated string, not including the terminating null. (Same as the function **strlen()**).

SYNOPSIS

int bytesize (string_address)
char *string_address;

DESCRIPTION

Return value: returns the number of bytes in the string.

NAME

gethostname – tills in the string descriptor with the current hostname. The hostname has a maximum of 31 characters.

SYNOPSIS

int gethostname(string_descriptor)
struct {short len, type; char *ptr;} *string_descriptor;

DESCRIPTION

Return value: If successful, returns a 0. If not successful, returns -1.

NAME

index – finds the first occurrence of a character in a null terminated string.

SYNOPSIS

```
char *index(string_ptr, search_char)
char *string_ptr;
char search_char;
```

DESCRIPTION

Return value: If the character is found, the address of the character is returned. It the character is not found, a 0 is returned.

NAME

lowercaseify – converts all uppercase characters A-Z in a null terminated string to lowercase. All other characters are ignored.

SYNOPSIS

int lowercaseify(string_ptr)
char *string_ptr;

DESCRIPTION

Return value: The number of characters scanned, not including the terminating null.

NAME

rindex – finds the last occurrence of a character in a null terminated string.

SYNOPSIS

char *rindex(string_ptr, search_char)
char *string_ptr;
search_char;

DESCRIPTION

Return value: If the character is found, the address of the character is returned. If no character is found, a 0 is returned.

NAME

uppercaseify – converts all lowercase characters a-z in a null terminated string to uppercase. All other characters are ignored.

SYNOPSIS

int uppercaseify(string_ptr)
char *string_ptr;

DESCRIPTION

Return value: The number of characters scanned is returned, not including the terminating null.

NAME

xclose – closes the channel/socket used tor XNS communications.

SYNOPSIS

int xclose(chan)
int chan;

DESCRIPTION

Return value: Returns a value of 0 if the channel is successfully closed. Returns a value of –1 if there is an error closing the channel. The VMS error return value is found in the external **vmserrno**.

NAME

xcmd – execute a remote command via XNS.

SYNOPSIS

xcmd (host, command)
char *host, *command;

DESCRIPTION

Xcmd makes a network connection on socket *EXECSOCKET*, on **host** and causes **command** to be executed remotely. It returns –1 if a connection can't be made. If the execution is successful, a channel is returned, to which the remote process is attached. Reads and writes to this channel will retrieve/send data from/to the remote process.

NAME

xcp - transfers tiles between hosts via Ethernet XNS.

SYNOPSIS

$ XCP :== $DEVICE:[IRIS.XNSN23]XCP !install as a foreign command
$! VMS to UNIX copy
$ XCP {-dv} filespec1 filespec2 ... hostname::pathname
$! UNIX to VMS copy
$ XCP {-bv} {hostname1::}pathname {hostname2::}pathname ... filespec

where:

DEVICE:[IRIS.XNSN32]
is the directory tor the XNS software.

{}
enclose optional entries.

pathname or filespec delimiters are spaces or tabs.

-b
is a flag that indicates a byte for byte copy from UNIX to VMS.

-d
is a flag that indicates the UNIX target pathname is a directory on the remote host.

-v is a flag that means be verbose.

hostnamex
is the name of a remote host machine.

filespec
is a VMS file specification.

pathname
is a UNIX pathname.

DESCRIPTION

The *XCP* utility is used to transfer tiles between hosts connected via Ethernet using XNS protocols. It is supported tor both UNIX and VMS systems, though VMS to VMS transfers are not yet supported.

Flags

-b is described in the section “VMS output files”,

-d is useful only for transfers to UNIX systems.

-v names the currently active file, and prints out a ". " for every 20k bytes transferred.

Case

Hostnames or UNIX pathnames are case sensitive, and are limited to 31 (decimal) alphanumeric characters or less. For example: "hostname", "HOSTNAME", and "Hostname" name 3 different hosts, "/foo", "/Foo", and "/FOO" are 3 different UNIX pathnames.

Because the *XCP* utility uses the VMS C compiler run-time library, all input from the command line is lowercased unless enclosed in quotes. For example:

```
$ XCP -V OLYMPUS::/USR/TMP/GARBAGE.TXT GARBAGE.TXT

$ xcp -v olympus::/usr/tmp/garbage.txt garbage.txt
```

will both copy the file /usr/tmp/garbage.txt from the remote host olympus to the file GARBAGE.TXT in the current directory.

```
$ XCP -V "olympus::/usr/tmp/GARBAGE.TXT" garbage.txt
```

will copy the file /usr/tmp/GARBAGE.TXT from the remote host olympus to the file GARBAGE.TXT in the current directory.

Legal Hosts

Currently only UNIX to VMS, or VMS to UNIX transfers are supported, hence, there must be one and only one hostname on the command line.

Wildcards

Wildcards in the filename, type, and version number are supported both locally and remotely (as the hosts would normally interpret them), and related input tile specifications are supported for VMS. Wildcards in the device or directory specification are not supported. For example, if the current directory holds A.MAR and B.MAR:

```
$ XCP -D *.MAR GINGER::/STAFF/ROGERS
$ XCP -D A.MAR B GINGER::/STAFF/ROGERS
```

will both copy A.MAR and B.MAR to the remote host "ginger" as /staff/rogers/a.mar and .staff/rogers/b.mar.

```
$ XCP GINGER::./STAFF/ROGERS/*.MAR SYS$DISK:[FRED]*. MAR
```

assuming "c.mar" and "d.mar" are in the directory ginger::/staff/rogers, this command will create the tiles SYS$DISK:[FRED]C.MAR and SYS$DISK:[FRED]D.MAR (using the latest version numbers).

Version Numbers

Version numbers may be included either through wildcards or specifically named. For example:

```
$ XCP -D BOGUS.DOC;1 FRED::/USR/TMP
```

will copy the file BOGUS.DOC;1to the UNIX host "fred" with a pathname of /usr/tmp/bogus.doc.1.

```
$ XCP -D *.MAR;* GINGER::/STAFF/ROGERS
```

Assuming the files A.MAR;2 and B.MAR;5 are in the current directory, this command will copy A.MAR;2 and B.MAR;5 to the remote host "ginger" as /staff/rogers/a.mar.2 and /staff/rogers/b.mar.5.

Filenames

Filenames are checked for correctness at their host. On VMS, related input file specifications are supported. Spaces or tabs are legal filename delimiters.

VMS Input Files

Only sequential files are currently supported. *XCP* uses the VMS C run-time library for emulating stream input from record files, of which the following is a summary from the VAX-11 C programming manual:

If the record attributes are implied carriage control (RAT = CR), then a newline is appended to the record.

If the record attributes are print carriage control (RAT = PRN), then the prefix and postfix carriage controls are expanded and concatenated before and after the record.

If the record attributes are FORTRAN carriage control (RAT = FTN), then the first byte of the record is removed, and prefix and postfix characters are concatenated to the record. The following rules describe the way the character in the first byte maps onto the prefix and postfix bytes that

appear in the emulated stream. <record> denotes the bytes contained in the logical record exclusive ot the first carriage-control byte.

\n denotes the newline character;

\f denotes the form-feed character;

\r denotes the carriage-return character;

```
NUL    -> <record>
0      -> \n\n<record>\r
1      -> \f<record>\r
+      ->   <record>\r
$      -> \a<record>
all others ->   <record>\r
```

CAUTION
An expanded record cannot exceed 512 bytes. Thus, the input record generally must not exceed 510 bytes of actual data, since up to two characters may be added in the expansion process.

VMS Output files:

Characteristics:
Sequential, contiguous best try
Extension quantity = default
Maximum version number

If the **-b** flag is used, then:
record format = fixed length (512 bytes)
record delimiter = none
record attributes = none

If the **-b** flag is NOT used, then:
record format = stream
record delimiter = line feed
record attributes = carriage return

Errors:

VMS gives up on the first error.

Bugs:

Transferring ASCII files from VMS to UNIX and back may result in the two files being a little different. Specifically, files are transferred over the network

in stream format, with <lf> record terminators. If the original file was (for example) an ASCII file using VFC records, then records could contain line feeds as valid characters. When converted to stream format, that record would then be converted to two records when sent over the net. They would then return as two records as well.

Restrictions:

Record sizes should not exceed the maximum record size for VMS, which is around 64K bytes. For ASCII file transfers, observe the 512 byte record size limit mentioned above. This version of XCP should be used with the current Silicon Graphics *IRIS Workstation Software Distribution, GL2-W2.3* or *IRIS Workstation Software Distribution, GL1-W2.3*.

NAME

xlogin – remote login

SYNOPSIS

xlogin [-txz] rhost

DESCRIPTION

Xlogin connects your terminal on the current local host system to the remote host system *rhost*. If no login is attempted within 60 seconds then *xlogin* times out. All echoing takes place at the remote site, so that *xlogin* is transparent except for network delays. Unless otherwise specified, all characters are immediately transmitted to the remote host system except ~. To disable ~ processing, use the **-t** flag.

If the **-t** flag is not specified and a ~ is typed immediately after a carriage return, then the ~ is not sent to the remote host and the next character is read. If the next character matches one of the following cases then the described action is performed, else the two characters are sent to the remote host:

~. Force *xlogin* to exit.

~% *Xlogin* reads another character. If the character is 'X' or Z' then local interpretation of special characters is toggled (see below). Otherwise, all three characters are sent to the remote host.

~! Escape to an interactive shell on the local system. The value of the environment variable SHELL determines which shell is executed. If *xlogin* finds the environment variable CMDNAME set, this local shell sets it to

xlogin:*hostname*

This is useful for altering shell prompts so that you are aware that you are in a sub-shell. Exiting the shell returns you to the remote host.

~~ Send a single '~'.

For the flow control characters (CTRL/S and CTRL/Q) and flush (CTRL/O) character to be interpreted locally on the calling machine, use the **-x** flag. Within *xlogin*, you can use the command, **~%X** , to toggle local and remote interpretation. When interpretation is done locally, the response is immediate rather than delayed and these special characters are not transmitted to the remote host. Note that programs which change the terminal characteristics and expect to receive these special characters (EMACS, for example) won't receive them when they are interpreted locally.

For local interpretation (i.e., the calling machine) of miscellaneous special characters, use the **-z** flag. The miscellaneous special characters currently supported are CTRL/V, CTRL/Y, and CTRL/Z under UNIX 4.2 BSD. Within *xlogin*, you can use the command, **~%Z** , to toggle local interpretation. Note

that programs which change the terminal characteristics and expect to receive these special characters (EMACS, tor example) won't receive them when they are interpreted locally.

SEE ALSO

xcp(1C), xx(1C)

BUGS

You cannot redirect the standard input or output of *xlogin*.

CTRL/O cannot be interpreted locally on VMS.

Escaping to an interactive shell does not work on VMS.

NAME

xnsconnect – creates a connection with "host" at the indicated socket number.

SYNOPSIS

int xnsconnect(host, socket)
char *host;
int socket;

DESCRIPTION

Xnsconnect returns a channel after making a *Sequenced Packet Protocol* (SPP) connection to the indicated **socket** at **host**. A –1 is returned on failure.

If any errors occur, the VMS error return value is found in the external **vmserrno**.

NAME

xnseof – sends a data stream end of tile message.

SYNOPSIS

int xnseof(chan)
int chan;

DESCRIPTION

Xnseof returns a 0 if successful. A -1 is returned on failure, and the VMS error code is left in the external **vmserrno**.

NAME

xnsfile – find an available network channel.

SYNOPSIS

int xnsfile()

DESCRIPTION

If successful, *xnsfile* returns a channel number.

A –1 is returned on failure, and the VMS error code is left in the external **vmserrno**.

NAME

xnslisten – listens on a socket using socket number 100<'socket'<2999.

SYNOPSIS

int xnlisten(socket)
int socket;

DESCRIPTION

Returns control to the program when a socket is available for reading or writing. Return value: If successful, returns the channel number. If not successful, returns -1. The VMS error return value is found in the external **vmserrno**.

NAME

xnsphysconnect – creates an SSP connection to a binary network address.

SYNOPSIS

```
typedef struct {
        unsigned short  high;
        unsigned short  low;
} Xnet;

typedef struct {
        unsigned short  high;
        unsigned short  mid;
        unsigned short  low;
} Xhost;

typedef unsigned short socket;

typedef struct {
        Xnet    net;
        Xhost   host;
        Xsocket socket;
} Xaddr;

struct xns_setup {
        Xhost physaddr;
        Xaddr internet;
        char name[NSIZE];




int xnsphysconnect(setup)
struct xns_setup *setup;
```

DESCRIPTION

Xnsphysconnect
returns a channel after making a Sequenced Packet Protocol (SPP) connection to the address indicated in the **xns_setup** structure. A -1 is returned on failure. If any errors occur, the VMS error return value is found in the external **vmserrno**.

NAME

xnsread – reads from a channel “chan” the “size” of bytes into the butter “buf”, with data type “dtype”.

SYNOPSIS

int xnsread(chan, buffer, size, dtype, control)
int chan, size;
char *buffer, *dtype, *control;

DESCRIPTION

Return value: If successful, returns the number of bytes read (0 to the end of file). If not successful, returns - 1. The VMS error return value is found in the external **vmserrno**.

NAME

xnswrite – write to XNS connection.

SYNOPSIS

xnswrite (chan, buf, count, dtype, control)
int chan, count;
char *buf, dtype, control;

DESCRIPTION

Xnswrite allows you to write to an XNS channel while specifying a particular data type and control bits. This data type can be recognized by the server, and can be used to pass out-of-band information, for example. A count of the number of bytes written is returned. The data type is reset to the default after each write. The VMS error return value is found in the external **vmserrno**.

NAME

xread – reads from a channel “chan” into the buffer “buf” the “size” ot bytes.

SYNOPSIS

int xread(chan, buf, size)
int chan, size;
char *buf;

DESCRIPTION

Return value: If successful, returns the number of bytes read (0 to the end of file). If not successful, returns -1. The VMS error return value is found in the external **vmserrno**.

NAME

xsh – start a shell via XNS.

SYNOPSIS

xsh (host)
char *host;

DESCRIPTION

Xsh makes a network connection on socket *XSHSOCKET* on **host** and forks a shell. It returns –1 if a connection can't be made.

NAME

xwrite – writes the “size” of bytes to the channel “chan” from the buffer “buf”.

SYNOPSIS

int xwrite(chan, buf, size)
int chan, size;
char *buf;
char *host;

DESCRIPTION

Return value: If successful, returns the number of bytes written. It not successful, returns -1. The VMS error return value is found in the external **vmserrno**.

NAME

xx – remote shell

SYNOPSIS

xx [-txz] host command

DESCRIPTION

Xx connects to the specified *host*, and executes the specified *command*. The remote login name must be equivalent (in the sense of *sh*(1)) to the originating account; no provision is made for specifying a password with a command. If you omit *command*, *xx* logs you into the remote host. In this mode, the switches [-txz] function the same as tor xlogin. See xlogin(1C) for switch meanings and character processing information.

If you do specify a command, the switches [-txz] have no effect. *Xx* copies its standard input to the remote command and the standard output and error of the remote command to its standard output. All echoing and special character handling takes place on the local machine. Thus, flow control characters (CTRL/S and CTRL/Q), and interrupt characters affect the *xx* process running on the local machine.

If *xx* is linked or copied to a file named *host* then executing *host* is the same as *xx host*.

Shell metacharacters which are not quoted are interpreted on the local machine, while quoted metacharacters are interpreted on the remote machine. Thus, the command

$ xx otherhost cat remotefile > > localfile

appends the remote file *remotefile* to the local file *localfile*, while

$ xx otherhost cat remotefile " > >" otherremotefile

appends *remotefile* to *otherremotefile*.

SEE ALSO

sgboot(1M), sgbounce(1M), xnsd(1M), utmp(3N), xcmd(3N), xconnect(3N), xnsfile(3N), xnsioctl(3N), xnslib(3N), xnswrite(3N), xlogin(1C), xcp(1C)

BUGS

If you are using *csh*(1) and put *xx* in the background without redirecting its input away from the terminal, it will block even if no reads are posted by the remote command.

You cannot run an interactive command (like *vi*(1)); use *xlogin*.

UNIX 4.2 BSD stop signals stop the local *xx* processes only; this is arguably wrong, but currently hard to fix for reasons too complicated to explain here.