Babel
Functional Specification

Distribution:	COMPANY CONFIDENTIAL
Title:	Babel Functional Specification
Drawing Number:	0384,020/FS
Issue:	E****DRAFT****
Author:	KSR
Date:	10.10.91
Change Number:	n/a
Last Issue:	D (DRAFT of 8.10.91)

Contents:

1. Project Babel overview

Project *Babel* will implement a networking system provisionally known as *Multi Network System (MNS).* Babel will bring the benefits of industry standard TCP/IP and Ethernet networking into the schools education sector.

MNS is a way of organising a TCP/IP based network which is suitable for users and network managers already familiar with our Econet offerings. It will initially enable non-disruptive migration away from Econet to Ethernet and will eventually form the basis of well-structured whole-school networks.

MNS will enable a new Ethernet segment to be linked transparently to an existing installed Econet, retaining the familiar Econet naming, addressing and user interface conventions across the whole expanded network. Similarly, MNS will enable departmental clusters based on Ethernet or Econet or some future hardware system such as cordless to link together via a backbone Ethernet. In the future, whole site networks may be able to link themselves together via MNS and appropriate wide-area communications technology.

The industry standard nature of Ethernet and TCP/IP will introduce concepts of Open Systems networking into the schools education sector. Over time, mixed networks of Archimedes, PCs, Apple and UNIX computers may develop as suitable applications for interworking are found.

MNS implementation will be founded upon TCP/IP software already developed for AES32 *TCP/IP Protocol Suite.* Some modifications to the *Internet module* component of AES32 will be required, but these modifications may be fed back into a future version of AES32, in order to maintain consistency between the two products. *TCP/IP Protocol Suite* includes applications, running over TCP/IP, which are suitable for Tertiary and VAS users. MNS will include applications, running over TCP/IP, which are suitable for school users. The two products will complement each other in our product range.

The MNS product package will include a version of Level 4 Fileserver software optimised for use over Ethernet. This will include !Server, !Manager and !Spooler.

MNS software will be licensed to third parties for inclusion in ROM on third party Ethernet cards for A3000 and other ARM-based platforms.

1.1 Future development

In the short term, many MNS installations will be small scale, consisting of just one or two Ethernet and Econet partitions on a single site. However project Babel will introduce the basic concepts and technology which will enable larger site-wide and multi-site network systems to develop over time. As interest in achieving modern network infrastructures in schools grows, so the market for support services, new applications and other third party products to maintain, exploit and expand that infrastructure will grow at the same time. An active but unpredictable marketplace for networking products and services in the schools education sector will emerge, with MNS as the catalyst.

2. MNS concepts

Functionally, an MNS network is a TCP/IP network containing a number of subnets linked by A5000 (or A420/A440) computers acting as gateways. However, it is the MNS veneer which will provide the unique selling proposition for our customers.

The way an MNS network is organised is reflected in the structure of an MNS network address. This is based upon a standard TCP/IP address but with an extra layer of meaning, derived from the requirements of our users in our core markets, superimposed. Users will normally address a network services by its name, rather than its address.

An MNS network address is a 4-byte quantity, expressed in the form:

site.partition.net.station

Examples: 1.1.3.2 1.127.127.6 2.1.6.12

The meaning of each byte in an MNS network address is as follows:

site A number of sites may be linked together to form a *site group*. This number identifies an individual site within a site group, and so must be unique within the group. A site group may contain between 1 and 127 sites. A site is connected to another site within a group via a *site gateway* and a suitable wide-area comms link.

partition A site contains one or more partitions. A partition contains one or more physical nets of the same type. Two partitions of different types (e.g Econet, Ethernet) may be linked together by an A5000 or A400 series computer functioning as a *partition gateway*. A partition gateway contains two hardware interfaces, connecting the machine to a net in each adjacent partition. IP messages are relayed between the two partitions via these interfaces.

The partition number identifies an individual partition. It needs to be unique only within the local site, not the entire site group. A site may contain up to 255 partitions. Partition numbers have arbitrary values between 1 and 255.

net This number identifies an individual physical net. Note that two Econets linked together by an Econet bridge will constitute two distinct **nets**, whereas two Ethernets linked together via an Ethernet bridge will consitute a single **net**. This is because two bridged Ethernets appears to users to be a single Ethernet whereas two bridged Econets remain distinct. Hence in an Ethernet based partition there will always be exactly one **net**, but in an Econet based partition there may be several **nets**.

The net number must be unique within an entire site group, NOT just the local site or the partition containing the net.

The net number is functionally equivalent to, and if the underlying net is an Econet **MUST** when possible (i.e when several nets are linked by Econet bridges) have the same actual value as, an Econet net number. Net numbers have arbitrary values between 1 and 255. although a numbering scheme to visibly distinguish Econet from Ethernet nets within the system may be recommended, e.g Econets in range 1 - 126, Ethernets in range 128 - 254, with the backbone network (a special case) as net number 127. Net numbers 0 and 255 are reserved.

station This number identifies a host station connected to a net. The number must be unique on that individual net. Station numbers have arbitrary values between 1 and 255. A partition gateway will have the same station number on both connected nets.

The MNS interpretation of an address in the form

site.partition.net.station

is

*"machine **net.station** , located within **site.partition**"*

For example, the MNS interpretation of a command - in the normal TCP/IP emphasis - to:

*"send data to host **1.3.129.16**"*

is actually:

*"send data to station **129.16** ... (which is located within partition number **3** in site number **1**)"*

or, more meaningfully:

*"send data to station **129.16** ... (which is located within partition **science** in site **StTriniansUpper**).*

The emphasis is on the field pairing **net.station.** Network applications need to display and handle only these two bytes (although most user interfaces will deal in service *names* which are then mapped transparently by software into a net.station address). This is identical to native Econet syntax. On physical Econets the numbers themselves will be the same when possible. The remaining MNS address fields **site.partition** represent additional routing information needed by the underlying TCP/IP software in order to route data to the destination physical network. MNS software will have learned, via previous transparent network exchanges with an MNS-configured L4 fileserver machine, the actual **site.partition** location of each accessible physical network, and so the user will not be required to supply this information. It will be filled in automatically by software.

2.1 MNS station address configuration

In line with Econet convention, the station number will be configured on each host computer via `!Setstation`. The remaining fields of the MNS address for each station will subsequently be constructed automatically by MNS software at startup time, dependant upon the relative location of the station within the MNS network at that time.

The semi-automatic address configuration of client stations is achieved via transparent communication with an L4 fileserver machine set up with MNS configuration information in file format. This MNS information will enable the fileserver to export on request site, partition and net numbers to client stations within its own partition. This is to enable each client to know its own unique full MNS address and also know the location of every **net** in the system.

2.2 Mapping MNS addresses to TCP/IP addresses

The mapping between an MNS address in the form

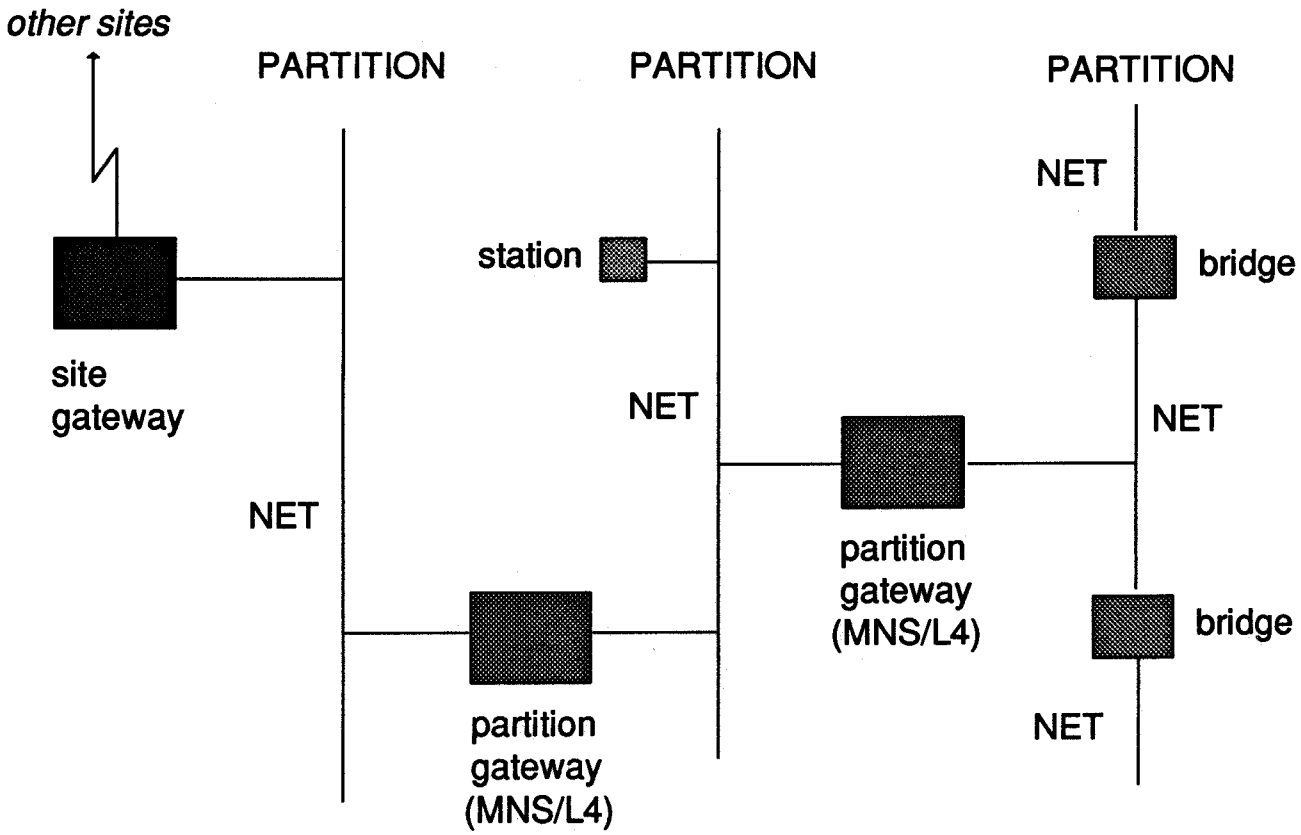
site.partition.net.station

and the standard interpretation of a Class A IP address is as follows:

i) The site number maps into a Class A IP network number.
ii) The partition number maps into a Class A IP subnetwork number (mask 0xffff0000).
iii) The two bytes net.station together map into a 16-bit Class A IP host identifier.

i.e. **net.subnet.host.host**

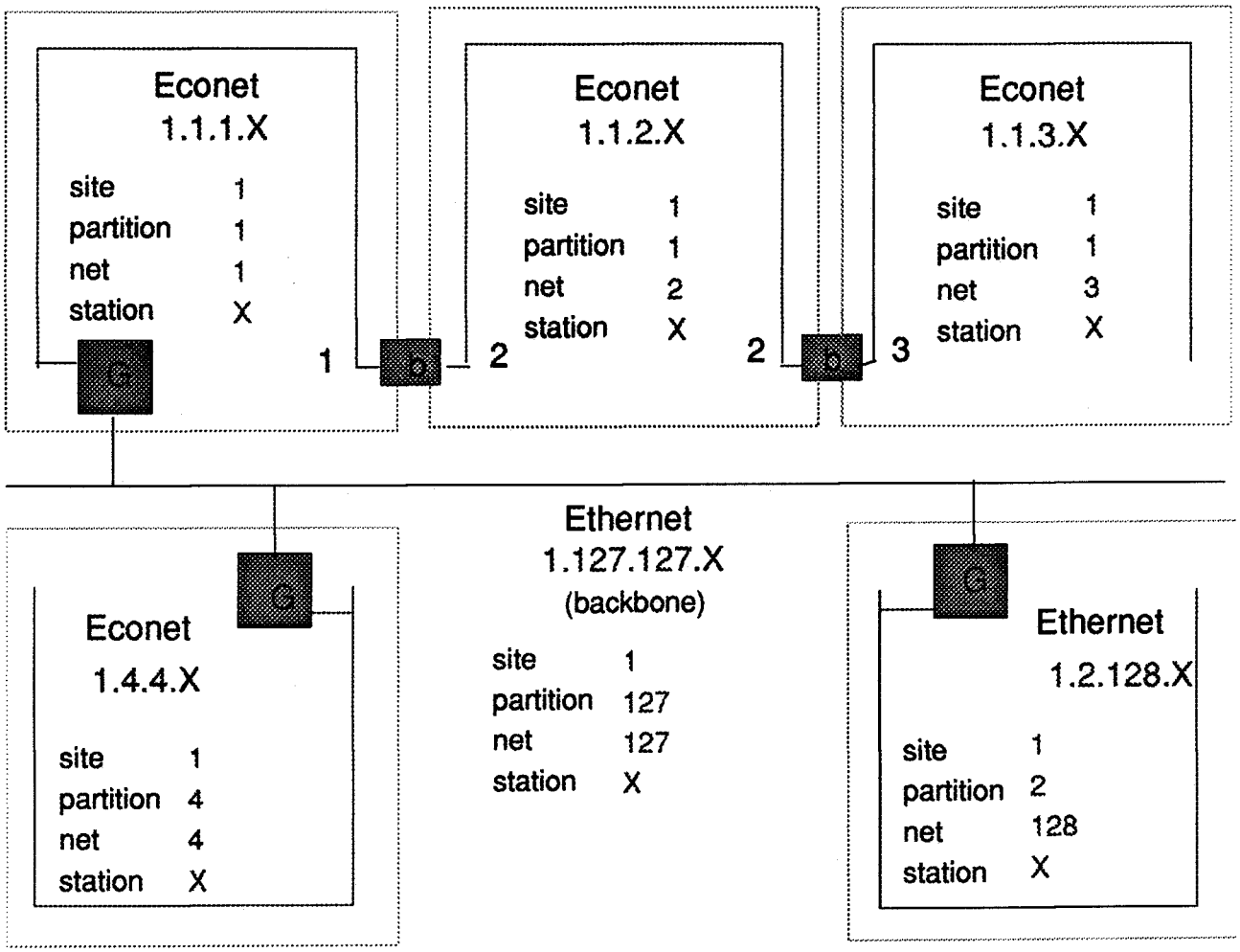
2.3 The MNS site model



2.4 Protocols

MNS is founded upon TCP/IP family protocols, including IP, ARP and Reverse ARP. The primary transport protocol will be UDP enhanced by an acknowledgement mechanism. TCP itself, as a stream oriented protocol, will not be required in this context.

2.5 Example



Note: In the above diagram, the various nets wil be visible to users, respectively, as

1.X
2.X
3.X
4.X
127.X (backbone)
128.X

3. MNS configuration

3.1 !SetStation

This is a desktop application similar to !Configure, to set operational parameters into CMOS RAM. Variables are:

net type

This variable selects the type of the local IP-based network. Options are **MNS** (default) or **STANDARD** (a standard TCP/IP network).

address configure

This variable sets the mode by which the local station obtains a full IP network address. Options are **LOCAL** or **REMOTE**. Remote means that a network server should be interrogated via Reverse ARP. This is the default mode. Local means that this information is stored in a local file and will be supplied by local software.

If net type is MNS then this variable must always be **REMOTE**.

If net type is STANDARD then this variable is **LOCAL** only if this software is being used in conjunction with Internet software components from *TCP/IP Protocol Suite* (which has its own configuration procedures via *!Internet.!Configure*), otherwise **REMOTE**.

device type

This variable sets the name of the hardware device used to access the network. This is a two or three byte text string assigned by the interface manufacturer. Current Acorn values are *"et"* (Ethernet 1), *"en"* (Ethernet 2) and *"eco"* (Econet). The special type *"file"* indicates that this information is supplied via a local file.

If net type is MNS then this variable
- i) is *file* if the station is an MNS-configured L4 fileserver or partition gateway, in which case the device parameters are specified in a CONFIGURE file on the server.
- ii) must be a real interface name in all client stations.

If net type is STANDARD then this variable
- i) is *file* if address configure is **LOCAL** (ref *!Internet.!Configure*).
- ii) must be a real interface name if address configure is **REMOTE**.

Station number

This variable sets the **station** number, in the range 1 - 255.

If net type is MNS then this variable
- i) sets the MNS station number, which must be unique on the immediate MNS *net*.
- ii) sets the Econet station number if the underlying network is Econet, to the same value.

Note: In MNS, a partition gateway will be connected to two partitions and hence two nets. Although the **net** numbers of the two nets will be different, the partition gateway must have the same **station** number (set here) on both nets.

If net type is STANDARD then this variable
- i) sets the Econet station number if the connected physical network is Econet.
- ii) is null if the connected physical network is Ethernet.

3.2 MNS server configuration files

Two MNS configuration files are kept in the `.MNS` subdirectory of the L4FS *!Server* application, viz

`<Server$Dir>.MNS`

Both are plain text files, created with a text editor (from templates included in the product) by the network manager. Both files must be kept secure.

3.2.1 MAP

The file `<Server$Dir>.MNS.MAP` contains the "map" of the entire MNS network. A copy of this file is kept in the MNS configuration directory of ***at least one*** L4 fileserver within each partition of the MNS network. Information extracted from this file enables the server machine to provide client stations within the connected partition (including other L4 fileservers not so configured) with their own unique full MNS addresses and also information about the relative locations of each net in the system for routing purposes. MAP file format is:

```
site
        .partition
                        .net
                        .net
                        .net
        .partition
                        .net    # comment
site
        .partition
                        .net    # etc..

# Example1: multi-site network containing several dept partitions + backbone,
#           incorporating multiple Econet and Ethernet nets.

StTriniansUpper
        .compsciA               # computer block Econet: 3 nets linked by bridges
                        .1      # 1.X   2.X  3.X
                        .2
                        .3
        .compsciB               # new compblock Ethernet 128.X
                        .128
        .science                # science block ethernet 129.X
                        .129
        .art                    # artroom econet 4.X
                        .4
        .business               # business studies dept ethernet 130.X
                        .130
        .127                    # 127.X is backbone interconnect ethernet
                        .127

StTriniansLower
        .science                # lower school science econet: 2 nets + bridge
                        .5      # 5.X  6.X
                        .6
```

The names of sites and partitions in the MAP file will be converted automatically into MNS site and partition numbers. Names are converted in read order, from the start of the file. If a number is provided instead of a name to identify a site or partition then that number will be used. Nets must be specified by number only.

In the example above:

```
name   StTriniansUpper                will generate site number        1
name   StTriniansLower                will generate site number        2

name   [StTriniansUpper].compsciA     will generate partition number   1
name   [StTriniansUpper].compsciB     will generate partition number   2
name   [StTriniansUpper].science      will generate partition number   3
name   [StTriniansUpper].art          will generate partition number   4
name   [StTriniansUpper].business     will generate partition number   5
number [StTriniansUpper].127          will generate partition number   127
name   [StTriniansLower].science      will generate partition number   1
```

Adding in the net numbers the following full MNS addresses will be generated, in order. The <station> field will be added in by each individual station, via its own configured !Setstation value:

```
1.1.1.<station>     # StTriniansUpper.compsciA.1.<station>
1.1.2.<station>     # StTriniansUpper.compsciA.2.<station>
1.1.3.<station>     # StTriniansUpper.compsciA.3.<station>
1.2.128.<station>   # StTriniansUpper.compsciB.128.<station>
1.3.129.<station>   # StTriniansUpper.science.129.<station>
1.4.4.<station>     # StTriniansUpper.art.4.<station>
1.5.130.<station>   # StTriniansUpper.business.130.<station>
1.127.127.<station># StTriniansUpper.127.127.<station>
2.1.5.<station>     # StTriniansLower.science.5.<station>
2.1.6.<station>     # StTriniansLower.science.6.<station>
```

Note: the above example also illustrates the rule that partition numbers need be unique only within its local site, whereas net numbers must be unique within the whole multi-site system.

```
# Example2: single-site network consisting of 1 Ethernet partition/net
DotheboysComp
        .comproom               # 128.X
                        .128
# Generated MNS addresses will be:
#           1.1.128.<station>



# Example3: single-site network consisting of a 1 Econet + 1 Ethernet partition.
BashStCTC
        .compstudiesA           # Econet 1.X
                        .1
        .compstudiesB           # Ethernet 128.X
                        .128
# Generated MNS addresses will be:
#           1.1.1.<station>
#           1.2.128.<station>
```

3.2.2 CONFIGURE

The file `<Server$Dir>.MNS.CONFIGURE` contains MNS configuration information for an individual L4 fileserver machine within the MNS network. Whereas the MAP file is the same throughout the system, this file will be unique on each MNS-configured L4 fileserver. The information in this file will:

a) enable the L4 fileserver to identify its own relative position within the complete MNS network, and so be able to disseminate correct MNS addresses to client stations.

b) enable underlying host TCP/IP software in the server to be configured to route messages if there are two hardware interfaces present, and so enable the L4 fileserver to also act as a *partition gateway* between adjacent MNS partitions.

File format is (# = comment):

```
# Site  is <sitename>
# Econet    is <partition_name> | <partition number>
# SlotN     is <partition_name> | <partition_number> <device_type>
#
# <sitename> and <partition_name> | <partition_number> appear in MAP
# <device_type> is manufacturer's name for the interface card (eg "en", "et")
#
# File contains two or three non-comment/white space fields (one per line>.
# If this host server is also a partition gateway then there will be
# three lines, otherwise two.

# Example1:
#    partition compsciA is Econet, partition backbone is Ethernet accessed
#    by an Acorn Ethernet II podule. Host will be a partition gateway.

Site    is StTriniansUpper
Econet  is compsciA
Slot0   is 127 en

# Example2:
#    partition compsciB is Ethernet accessed by an Acorn Ethernet I podule,
#    partition backbone is Ethernet accessed by an Acorn Ethernet II podule.
#    Host will be a partition gateway.

Site    is StTriniansUpper
Slot0   is compsciB et
Slot1   is 127 en

# Example3:
#    partition comproom is Ethernet accessed by an Acorn Ethernet II podule.
#    this is a simple MNS comfiguration consisting of a single site containing a
#    single partition. Host will not need to act as a partition gateway.

Site    is DotheboysComp
Slot0   is comproom en
```

4. MNS client station startup

Once loaded and running over the configured network interface device, the MNS software must:

a) initialise the hardware interface to be used to connect to the network. This is done after reading the MNS device type previously configured into CMOS RAM via `!SetStation`.

b) discover its own full MNS (IP) address. This will be done via a Reverse ARP *request/response* exchange with a local MNS-configured L4 fileserver, with the client supplying a 6 byte pseudo-Ethernet address in the form **station:net:0:0:0:0** and the server returning a 4 octet IP address in the MNS format **site.partition.net.station.**

c) obtain information derived from a local MNS-configured L4 fileserver's MAP file which enables the client to map user-supplied **net.station** addresses into four byte MNS addresses in the form **site.partition.net.station,** and hence into standard IP addresses in the form **net.subnet.host.host.** This information is obtained via a second exchsnge of messages with the local L4 fileserver, via *Address Transform Protocol*, described in Appendix D.

d) obtain IP routing tables from which enable IP datagrams to be routed within the network. Routing within an MNS network conforms to standard Internet conventions; routing tables are obtained via *RIP (routed)* transactions.

5. MNS published interfaces

5.1 Application program interface

The API to an MNS network is a functional superset of the Econet SWI interface. Existing user applications which access Econet will not require modification at the network interface apart from textual editing of SWI names as follows:

Econet_CreateReceive	=>	**MNS_CreateReceive**	**(SWI &43d40)**
Econet_ExamineReceive	=>	**MNS_ExamineReceive**	**(SWI &43d41)**
Econet_ReadReceive	=>	**MNS_ReadReceive**	**(SWI &43d42)**
Econet_AbandonReceive	=>	**MNS_AbandonReceive**	**(SWI &43d43)**
Econet_WaitForReception	=>	**MNS_WaitForReception**	**(SWI &43d44)**
Econet_EnumerateReceive	=>	**MNS_EnumerateReceive**	**(SWI &43d45)**
Econet_StartTransmit	=>	**MNS_StartTransmit**	**(SWI &43d46)**
Econet_PollTransmit	=>	**MNS_PollTransmit**	**(SWI &43d47)**
Econet_AbandonTransmit	=>	**MNS_AbandonTransmit**	**(SWI &43d48)**
Econet_DoTransmit	=>	**MNS_DoTransmit**	**(SWI &43d49)**
Econet_ReadLocalStationAndNet	=>	**MNS_ReadLocalStationAndNet**	**(SWI &43d4a)**
Econet_ConvertStatusToString	=>	**MNS_ConvertStatusToString**	**(SWI &43d4b)**
Econet_ConvertStatusToError	=>	**MNS_ConvertStatusToError**	**(SWI &43d4c)**
Econet_ReadProtection	=>	**MNS_ReadProtection**	**(SWI &43d4d)**
Econet_SetProtection	=>	**MNS_SetProtection**	**(SWI &43d4e)**
Econet_ReadStationNumber	=>	**MNS_ReadStationNumber**	**(SWI &43d4f)**
Econet_PrintBanner	=>	**MNS_PrintBanner**	**(SWI &43d50)**
Econet_ReleasePort	=>	**MNS_ReleasePort**	**(SWI &43d52)**
Econet_AllocatePort	=>	**MNS_AllocatePort**	**(SWI &43d53)**
Econet_DeAllocatePort	=>	**MNS_DeAllocatePort**	**(SWI &43d54)**
Econet_ClaimPort	=>	**MNS_ClaimPort**	**(SWI &43d55)**
Econet_StartImmediate	=>	**MNS_StartImmediate**	**(SWI &43d56)**
Econet_DoImmediate	=>	**MNS_DoImmediate**	**(SWI &43d57)**

The MNS SWI set also includes one extra call with no functional equivalent in Econet:

MNS_ReadTransportType (SWI &43d58)

Reads whether a remote station must be accessed via UDP/IP or raw Econet protocols.

On entry
R0 = station number
R1 = network number

On exit
R0 = station number
R1 = network number
R2 = type value

Use
This call enquires whether the given station must be accessed via UDP/IP protoccols, e.g an MNS-configured L4 fileserver connected to Ethernet, or via raw Econet, e.g a Filestore or an L4 fileserver connected to the same Econet. On exit, R2 has the following value:

1 UDP/IP
2 Econet

This call enables user software to optimise variable parameters such as buffer sizes or count and delay values for individual I/O operations, depending upon the type of the underlying transport service used to reach the destination station.

An MNS support module will receive MNS SWI calls from user software and - if the destination station must be accessed via UDP/IP - map the commands into *socket interface* calls to the Internet module having expanded the associated two octet **net.station** address parameters into four octet **site.partition.net.station** MNS (IP) addresses. If the local partition is Econet and the destination station is also connected to this partition then the MNS SWI calls will always be mapped directly into the equivalent Econet SWI calls (even if the destination station is MNS-configured) to avoid unnecessary UDP/IP protocol overheads for this localised transaction.

When transmitting to a station via UDP/IP, *MNS_PollTransmit* and *MNS_DoTransmit* will return only the error values *Status_NetError* and *Status_NotListening* in the event of failure. When trasmitting over raw Econet other Econet-specific error values may be returned. MNS data transfer procedures are described in Appendix D.

5.1.1 Immediate operations and events

MNS will support a set of immediate operatons which are functionally the same as the equivalent set of Econet immediate operations. Argument syntax and semantics are the same, including the set of *OS_Procedure* calls, except that the prefix is *MNS_* . Procedures for performing immediate operations over UDP/IP are described in Appendix D.

1 **MNS_Peek**
2 **MNS_Poke**
3 **MNS_JSR**
4 **MNS_UserProcedureCall**
5 **MNS_OSProcedureCall**
6 **MNS_Halt**
7 **MNS_Continue**
8 **MNS_MachinePeek**
9 **MNS_GetRegisters**

MNS will share four event numbers with Econet, *Event_MNS_UserRPC (8)* and *Event_MNS_OSProc(16)*, *Event_MNS_Rx (14)* and *Event_MNS_Tx (15)*.

Reference: RISC OS Programmer's Reference Manual (for functional details of equivalent Econet operations)

5.2 Driver control interface

A software interface will control the flow of control information and data passing between a device driver module and a user protocol module, in this case the *Internet* module.

Reference: Appendix B.

6. NetFS and NetPrint

MNS client services are functionally the same as existing Econet applications, i.e NetFS and NetPrint. Configuration, user and program interfaces to these services are the same as to current versions. Internally the NetFS and NetPrint will be modified to use *MNS_ReadTransportType* and optimise individual transmissions for either UDP/IP or Econet.

References: Archimedes User Guide
RISC OS Programmer's Reference Manual

7. Level 4 Fileserver

The file and print server functionality of the L4 fileserver software included with MNS product will be the same as the Econet-only L4FS product which is current at time of launch. This includes *!Spooler*, *!Manager* and *!Server* components. The software will be modified internally for optimum performance over Ethernet via UDP/IP, and small display alterations will be made to distinguish the MNS version visually from the Econet-only version.

Reference: Level 4 Fileserver Functional Specification

In an MNS network, an L4 fileserver will often also be configured to provide MNS services as well, such as disseminating client station addresses and address-map information and/or acting as a gateway between two adjacent partitions. This enhanced service is reflected in the way that product will be sold, via a combined MNS / L4FS software pack. A new subdirectory within the L4FS *!Spooler* application directory, called `.MNS`, will contain MNS configuration files as described in Section 3 of this document.

8. Mixing MNS and non-MNS stations

Client stations configured with MNS software will be able to connect to the same Econet as stations not configured with MNS software, such as BBC and Master 128 computers.

Client stations configured with MNS will be able to access MNS-configured machines in all other MNS partitions via partition gateways and IP messages. Stations not configured with MNS will be able to access machines only in the local Econet partition via raw Econet messages. The names and addresses of machines on the local Econet will be the same to users of both types of client stations.

Client stations connected to Ethernet MUST be configured with MNS software.

8.1 Fileserver access

Client stations configured with MNS on an Econet will be able to access non-MNS fileservers, such as Level 3 fileservers and Filestores, connected to the same Econet. Client stations without MNS will also be able to access MNS-configured L4 fileservers connected to the same Econet.

9. Connecting other computers to an MNS network

An MNS network is a *real* TCP/IP network. The MNS addresses generated from a server's MAP file are valid IP addresses and the overall network represents a valid Class A TCP/IP site network with subnets. Routing within the system follows standard IP routing conventions.

Therefore other types of computer which support TCP/IP, such as PCs, Apple or UNIX machines, will be able to connect into an Ethernet partition of an MNS TCP/IP network - but must be configured with full IP addresses in line with those auto-generated by MNS software (see configuration examples in Section 3). These other types of computer will not support Acorn NetFS and NetPrint protocols and so will not be able to access L4 fileservers, but will be able to run "standard" application-level protocols (eg NFS, Telnet) to communicate with each other on top of the MNS TCP/IP transport base. (*Note:* A RISC OS computer configured with *TCP/IP Protocol Suite* software would be able to communicate with a UNIX computer over an MNS network).

10. Platform/OS support

MNS software will run on any Acorn computer with 1Mbyte RAM or more, under RISC OS 2.00 or later. A local hard or floppy disc will not be required on client stations when the MNS software is loaded out of an Ethernet podule ROM. A local disc will be required on an MNS-configured L4 fileserver / partition gateway.

11. Outline development/test strategy

MNS and TCP/IP components will be written in 'C' language, using ANSI C Release 4 in the Desktop C environment. Existing NetFS and NetPrint modules are written in ARM assembler.

A small standalone test network consisting of short segments of Econet and Ethernet and a small number of A3000 A400 and A5000 stations will be the main development base. Alpha testing will be done on the internal Acorn network. The beta test phase will involve a number of external school test sites, selected and monitored in association with EBU, but including at least one participant in the Educational Link Scheme. An important requirement for the beta test phase is to ensure that external test sites are able to come "on-line" as quickly as possible, with an adequate number of participating machines. A number of Ethernet 2 interface cards and possibly also a number of A420 or A440 computers to act as test stations may need to be requisitioned as loan stock for beta test sites, or else offered to beta test sites at reduced cost as an incentive to participate.

12. Product organisation and packaging

The major product deliverable is a combined MNS / L4FS software pack.

Slip case: "ACORN SOFTWARE"
Slip case inner box: "MULTI NETWORK SYSTEM / LEVEL 4 FILESERVER"

Level 4 Fileserver Network Manager's Guide
Acorn Multi Network System Handbook

Release note Registration document Site license agreement

2 X L4FS floppy discs (optimised for Ethernet and including MNS configuration files and software components)
1 X MNS station configuration floppy disc (MNS client station software components, !SetStation)

12.1 Manual style and contents

12.1.1 Level 4 Fileserver Network Manager's Guide

This manual will follow the same format as the current version, but updated textually to reflect use over an MNS network rather than raw Econet alone. A brief extra section will describe in overview the additional MNS subdirectory within the L4FS *!Spooler* application directory, but will refer the reader to the Acorn MNS Handbook for a detailed description of the files within.

12.1.2 Acorn Multi Network System Handbook

This manual will be a tutorial description of Acorn Multi Network System, structured to reach not only the technically aware school network manager but also the less technically aware school administrator. The handbook will present the ideas and principles underpinning MNS as well as guidance about laying out a school MNS network and instruction about configuring the various parts of the system. Some practical guidance about the use and maintenance of both Ethernet and Econet within an MNS network will be provided, with advice about how to expand the system over time.

12.2 Artwork design approach

Artwork on the inner slip case will include design motifs from the current Level 4 Fileserver product inner slip case combined with new design motifs illustrating the extra MNS functionality within this product.

Artwork on the Level 4 Fileserver Network Manager's Guide will be the same as the current version.

Artwork on the Acorn MNS Handbook will feature the same new MNS design motifs as will appear on the inner slip case.

12.3 ROM products

A podule ROM for Ethernet II will be produced, containing a driver module, MNS modules, the Internet module and the optimised versions of NetFS and NetPrint. This will be sold with the card, in association with the main MNS/L4FS software/manual pack via a site licensing arrangement. A further licensing arrangement will enable third party developers to ship their own Ethernet interfaces with the same software components already located in on-board ROM. This will ensure that third party cards can link properly into an installed MNS network.

A demonstration MNS version of the RISC OS 3 ROM will be produced for evaluation, in EPROM. In this version the RISC OS Application suite will be replaced with MNS software, as network users can down-load the displaced applications from network fileservers.

13. External dependencies

13.1 Low cost Ethernet cards

The availability of low cost (100 - 150 pounds REP) third party Ethernet cards for our ARM-based platforms at launch time will greatly increase the chances of success for MNS in the schools. Minimum requirement will be support for 10BASE2 ("cheapernet" or "thin wire" Ethernet) plus a podule ROM to hold the MNS software. 10BASE2, with a low cable cost and segment length of 185 metres will be the main Ethernet variant in local room or departmental

partitions, which is where most end-user machines will be concentrated. Longer backbone partitions, for example backbones which need to connect up different buildings, may be 10BASE5 ("thick wire") Ethernet. This has a higher cable cost but is physically rugged and can support segment lengths up to 500 metres. Short, localised backbone partitions may be based upon 10BASE2 Ethernet.

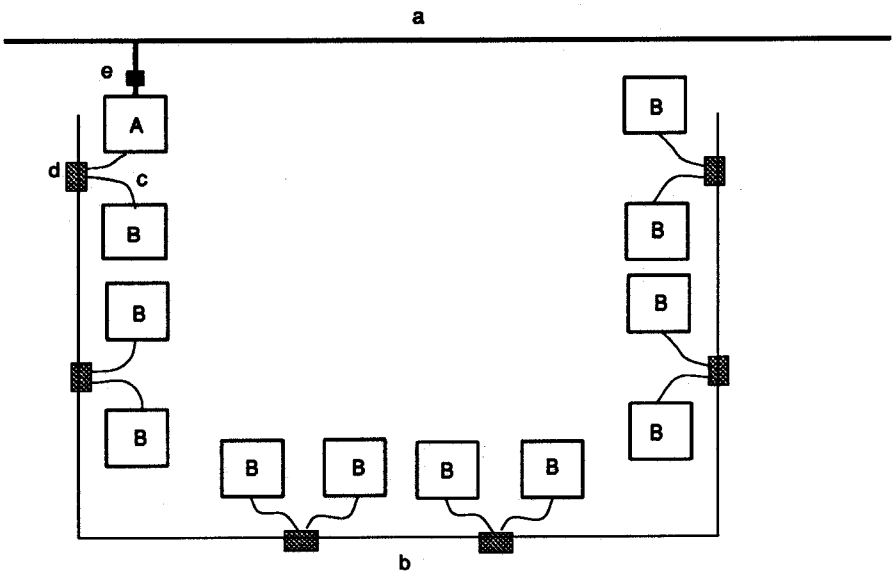
Note: Acorn's Ethernet II podule supports both 10BASE5 and 10BASE2. Hence this product would complement third party Ethernet cards in the marketplace if the latter support only 10BASE2 for cost reasons.

13.2 Ethernet infrastructure

The basic structure of an MNS site network is one of physically distinct subnets or *partitions*, probably often functionally and geographically associated with a particular room, department or curriculum area, which are interlinked via A5000 or A420/A440 *gateway* computers. This structure means that relatively expensive Ethernet infrastructure hardware - repeaters and bridges - will not often be *needed* to create a site-wide MNS system. However the MNS model will accommodate them when they are necessary, for example in a complex backbone partition.

13.3 Cheapernet partitions

The best way of installing Cheapernet in the classroom is to use a *Thin Wire Ethernet Port* system, manufactured either by Acorn or by a third party Ethernet equipment supplier [perhaps badged by Acorn]. This system features cable concealed in trunking around the walls, with socket box style outlet ports positioned at intervals. Stations are connected to the outlet ports by drop leads; cconnecting and disconnecting a staion does not interrupt the signal path The following diagram illustrates a classroom-sized MNS partition configured in this way. [This type and size of partition installation might usefully be offered by Acorn as a single packaged product similar to an Econet cluster.]



KEY

a thick wire Ethernet (MNS backbone partition)
b thin wire Ethernet (MNS department partition)
c thin wire Ethernet port connecting cable
d thin wire Ethernet port outlet box
e thick wire Ethernet drop cable + transceiver

A MNS-configured L4 fileserver and partition gateway
B Cheapernet-hosted client station

Note: the backbone partition may also be thin wire Ethernet

Cheapernet partition layout

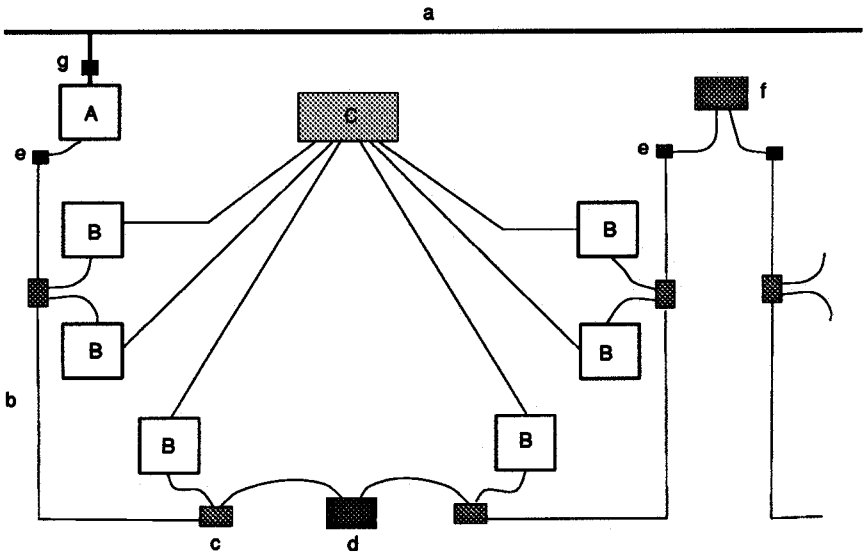
13.4 Backbone Ethernet partitions

The precise layout of a backbone partition, which will usually interlink departmental partitions, will depend upon the architectural plan of each individual site and the desired extent of the integrated network within the site. In large installations the backbone partition may, for example, call for a single long "snake" of 10BASE5 Ethernet cable or perhaps a more complex structure involving risers, multiport repeaters and multiple Cheapernet segments. Small installations may require no more than a short single segment of Cheapernet.

Correct design and installation of the backbone partition will be crucial to the success of a whole site network. Collaboration between Acorn (or its dealers) and a third party company offering specialist Ethernet design services may be beneficial in this case.

13.5 Econet partitions

Econet partitions will be configured in the same way as existing Econet clusters, with or without optional local application servers such as Nexus or SCSIShare, or management tools such as ClassROM. The same principles which now apply to clusters will also apply to Econet-based MNS partitions, in terms of careful use of bridges and limitations on the number of stations connected to each Econet segment. An existing cluster may be integrated easily into a developing MNS environment (perhaps formed by the installation of a new Ethernet backbone to interlink existing departmental nets). An existing L4 cluster fileserver may be upgraded to become an MNS partition gateway by the addition of an Ethernet card.



KEY

a thick wire Ethernet (MNS backbone partition)
b Econet (MNS department partition)
c Econet socket box
d Econet clock box
e Econet terminator
f bridge to adjoining Econet
g thick wire Ethernet drop cable + transceiver

A MNS-configured L4 fileserver and partition gateway
B Econet-hosted client station
C Application server (Nexus / SCSIShare) - OPTIONAL

Note: the backbone partition may also be thin wire Ethernet

Econet partition layout

Appendix A

L4 fileserver access via a standard TCP/IP network

A RISC OS user may sometimes want to access a L4 fileserver across an Ethernet-based standard TCP/IP network such as Acorn's own internal network, or a University campus network. Such a network will not be structured according to MNS rules, and so the MNS conventions which allow two byte *net.station* addresses to be mapped easily into 4 byte IP addresses will not be available. Required destination IP addresses may be in any address class format.

To achieve this, the user must configure the net type variable in !SetStation to STANDARD. This causes the MNS software to employ the following alternative address transformation mechanism.

The standard four byte IP address of the destination L4 fileserver is mapped into a two byte *pseudo* net.station address, with the special network number 255. This is the address seen by users and handled by NetFS and L4FS software. The list of address mappings is stored on L4 fileservers in the file `<Stations$Dir>.STATIONS` and is disseminated to client RISC OS stations via protocol exchanges described in Appendix D.

Note: Each client station may own its own local version of `<Stations$Dir>.STATIONS`. This may contain a a personalised set of address mappings to complement or override mappings obtained from network servers.

The STATIONS file format is (# = comment):

```
#
# 255.station       a.b.c.d
#
# Examples:

255.1       89.0.2.31
255.2       89.0.2.64
255.3       88.0.2.16
255.4       128.3.16.56
```

Appendix B

Driver Control Interface

This appendix describes the interface between a protocol module and a network device driver module used by the protocol. The interface will enable multiple protocol stacks to control multiple different device drivers simultaneously, if required.

The interface is optimised in its detail to handle device drivers for Ethernet, which will be the core LAN type in MNS. Drivers for other types of network will need to emulate Ethernet in these details at this interface and map "virtual Ethernet" values into real values meaningful to the actual connected network.

Specifically:

i) physical network addresses are 48 bit quantities.

ii) the values of physical network frame types "owned" by protocol modules are expressed as Ethernet frame type values. Example:

> *Internet* owns three types of physical frame, namely frames containing *IP*, *ARP* and *Reverse ARP* protocol messages. The driver module will be informed via the values *&800*, *&806* and *&8035* respectively.

B.1 Service calls

When loaded, a network interface driver module will perform various internal and hardware initialisation functions. It will then wait for a protocol module, such as *Internet*, to search for driver modules controlling network interfaces which the protocol wishes to access. This is done via the service call *Service_FindNetworkDriver*. Subsequently, if a protocol module terminates it will notify associated driver modules via *Service_ProtocolDying*.

Service_FindNetworkDriver (Service Call &84)

On entry
R1 = &84 (reason code)
R2 = pointer to name of network driver sought ("et", "en", etc)
R3 = pointer to **Protocol Information Block** describing this protocol

On exit
All registers preserved (if not claimed)

If claimed:
R1 = 0
R2 preserved
R3 = pointer to **Driver Information Block** describing available driver module

Use
Service_FindNetworkDriver makes a logical connection between a protocol module and a driver module, enabling information about each other to be exchanged.

Service_ProtocolDying (Service Call &83)

On entry R1 = &83 (reason code)
R2 = ID of exiting protocol

On exit All registers preserved to pass on

Use *Service_ProtocolDying* is issued by a protocol module to notify driver modules that the protocol is exiting. The protocol ID is the same value as previously passed to the driver in pib.pib_sccall. Driver modules must never claim this service call.

Protocol Information Block

```
struct pib {
        char            *pib_name;
        char            pib_frtypecnt;
        unsigned short  pib_frtype[6];
        int             pib_rxevent;
        struct mbuf     **pib_freeq;
        int             pib_sccall;
};
```

pib_name Pointer to name of protocol. *Internet* = "tcp_ip"

pib_frtypecnt Number of valid fields in pib_frtype[].

pib_frtype[] Array of physical frame type values which are "owned" by this protocol. The driver module will route all incoming frames with these types to this module, via an event sequence governed by pib_rxevent.

pib_rxevent Number of RISC OS event to generate when an incoming frame of the correct type is received. This mechanism is used to pass incoming frames into the correct protocol module.

pib_freeq Address of free list of empty data buffers owned by this protocol module but available to the driver module to store data associated with incoming frames of the correct type.
[*Note:* driver modules must never themselves free data buffers obtained from this list.]

pib_sccall ID for this protocol which will be included in Service_ProtocolDying on module termination. *Internet* = 1.

Driver Information Block

```
struct dib {
        char            *dib_name;
        int             dib_units;
        int             dib_swibase;
        char            *dib_address[4];
};
```

dib_name Pointer to text string name of physical interface type controlled by this driver module (e.g "en").

dib_units Number of accessible physical interfaces present of the type controlled by this driver module.

dib_swibase Base of SWI block owned by this driver module.

dib_address[] Pointers to physical addresses of interface cards. Each address is a 48 bit quantity.
[*Note:* the array size 4 represents a pragmatic limit.]

Once the startup sequence is complete, the protocol module will communicate with the driver module via SWI calls, and the driver module will interrupt the protocol module with events to indicate received frames.

B.2 SWI calls

The following set of SWI calls enable a protocol module to pass data and control commands to a device driver module. Each different driver will own a unique chunk of SWI numbers whose base is passed to a protocol module at startup time via the Driver Information Block. SWI numbers offset sequentially from the SWI chunk base will correspond functionally to the commands described below.

DCI_NetworkIfStart (SWI &(dib_swibase + 0))

Start a physical interface unit controlled by the driver module owning *dib_swibase*.

On entry R0 = interface unit number (0 - 3)
On exit Registers preserved
Use Called by protocol module to start interface and enable subsequent I/O.

DCI_NetworkIfUp (SWI &(dib_swibase + 1))

Restart a physical interface unit.

On entry R0 = unit number (0 - 3)

On exit Registers preserved

Use Called by protocol module to restart interface and reenable subsequent I/O, after an earlier call of **SWI_NetworkIfDown.**

DCI_NetworkIfDown (SWI &(dib_swibase + 2))

Disable a physical interface unit.

On entry	R0 = unit number (0 - 3)
On exit	Registers preserved
Use	Called by protocol module to disable indicated interface and disallow subsequent I/O.

DCI_NetworkIfSend(SWI &(dib_swibase + 3))

Transmit data via a physical interface unit.

On entry	R1 = unit number (0 - 3) R2 = frame type R3 = pointer to destination physical address: 48 bit (6 byte) quantity R4 = pointer to data buffer chain for transmission R5 = event number to call on completion or error (or zero for no event required)
On exit	Registers preserved
Use	Called by protocol module to transmit data, held in a chain of buffers. The destination physical address and a frame type value identifying the sending protocol are specified. If the protocol module wishes to be notified about the status of the transmission (beyond any error value which may be passed back directly on SWI exit) then the event number (> 0) will be specified.

B.3 Events

An event may be generated by a driver module to indicate that a data transmission request has been processed or that a frame has been received from the network. Different event numbers are owned by different protocol modules, and are supplied to driver modules via **SWI_NetworkIfSend** (for tx) and **Service_FindNetworkDriver** (for rx) calls.

TX Event

On entry to event handler	R0 = tx event number (specified by protocol module) R1 = pointer to data buffer chain containing tx data R2 = pointer to name of interface controlled by this driver ("et", "en", etc) R3 = physical unit number (0 - 3) R4 = error number (driver specific) or zero = ok

A transmission event does not necessarily imply that a frame has been successfully transmitted and received by the target host, merely that the local operation has been completed - either with or without a detected hardware error - and so the protocol module may free the associated data buffer chain. A protocol module has the option of requesting an event or no event with each SWI call to transmit data.

RX Event

On entry to event handler
R0 = rx event number (protocol specified)
R1 = pointer to data buffer chain containing rx data
R2 = pointer to name of interface controlled by this driver ("et", "en", etc)
R3 = physical unit number (0 - 3)
R4 = rx frame type

A receive event means that an incoming frame "addressed" (via the frame type field) to a protocol module has been received and stored within the addressed data buffer chain obtained for this purpose from the protocol module's freelist. Once the event is generated, the driver module must forget about the associated buffer chain. It will be received by the protocol module's event handler and in due course returned by the protocol module to its own freelist.

The first mbuf in the chain does not contain frame data. The first four bytes contains a pointer to a Driver Information Block describing this driver. The next six bytes contain the 48-bit physical address of the source of the received frame.

B.4 Data buffers

Data passes across the interface between the protocol and driver modules via structures called *mbufs*. These are the same data structures as used internally within the BSD UNIX kernel, and also within the RISC OS *Internet* module, for handling network data. The procedures for manipulating mbuf chains are also the same. Each mbuf is 128 bytes in size and can store internally up to 112 data bytes. The basic format is:

```
struct mbuf {
        struct mbuf     *m_next;        /* mbuf chain pointer */
        u_long          m_offset;       /* offset from start of mbuf to start of
                                           active data within m_dat[] */
        short           m_len;          /* length of active data in m_dat[] */
        short           m_type;         /* not used by driver module  */
        u_char          m_dat[112];     /* data storage area */
        struct  mbuf    *m_act;         /* not used by driver module */
```

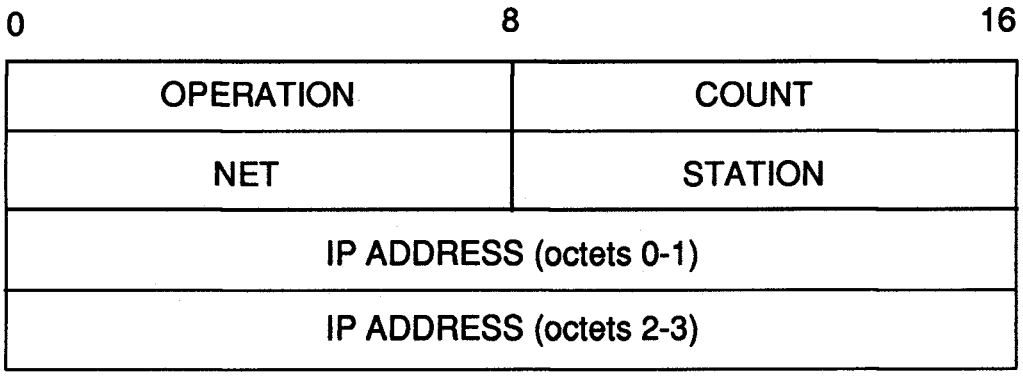
Appendix C

Address Transform Protocol

Address Transform Protocol (ATP) is an Acorn *client request / server response* protocol running over UDP. It enables a client station to request a list of address pairings, which will enable the client station subsequently to map any user-supplied net.station address into a full four byte address known to the underlying IP network. The four byte address will be in MNS format (i.e IP Class A) if the local network is an MNS network, or in standard IP format (i.e any IP Class) if the local network is a standard TCP/IP network.

The UDP port number used by ATP is &8100.

C.1 ATP Message format



ATP Message Format

Field *OPERATION* specifies an *mns_transform_request (1)*, an *mns_transform_response (2)*, a *standard_transform_request (3)* or a *standard_transform_response (4)*.

Field *COUNT* indicates the number of *ADDRESS BLOCKS* in the following list. This will be zero in a request message, a positive number in a response. Each address block is a six byte quantity. The first two bytes give a **net.station** pair; the following four bytes give the corresponding IP address, either in MNS or standard IP format depending upon the operation type.

C.2 mns_transform

In this mode a client station on an MNS network can obtain an address list derived from an MNS-configured L4 fileserver's MAP file which enables it to map user-supplied two byte **net.station** addresses into four byte MNS addresses, adding the additional **site.partition** information, e.g

129.16 => 1.3.129.16

The list of addresses specifies the address of every net within the local MNS system. The **station** octet is not relevant in this procedure and is set to zero in both parts of the address block.

C.3 standard_transform

In this mode a client machine on a standard TCP/IP network can obtain an address list derived from an L4 fileserver's STATIONS file which enables the client to map *specific* user-supplied net.station addresses into *specific* four byte standard IP addresses. These specific addresses will usually correspond to L4 fileservers connected to the network. E.g:

255.1 => 89.0.2.31

The net address 255 is reserved for this facility, which means that up to 255 direct mappings are possible by this mechanism. The station number is a logical number which does not necessarily correspond to a configured physical Econet station number on the addressed machine.

Appendix D

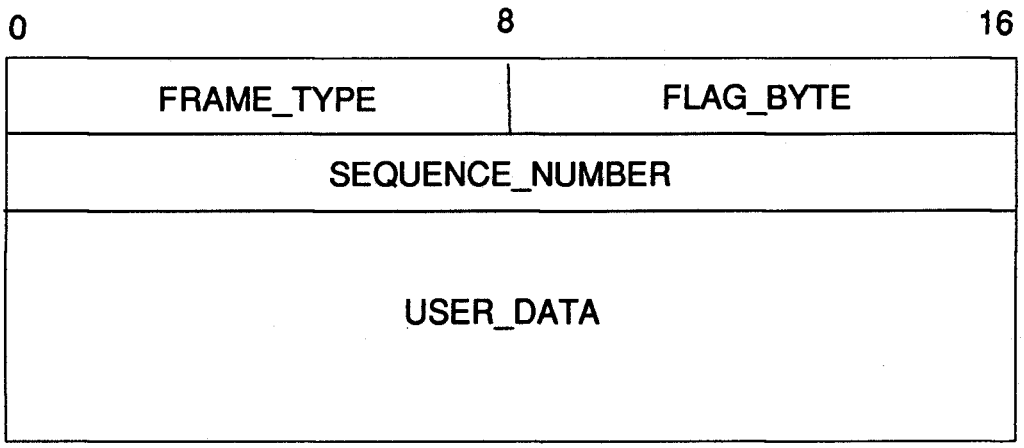
MNS frame formats and data transfer procedures

D.1 Data transfer over raw Econet

MNS_StartTransmit and *MNS_DoTransmit* SWI calls map directly into the equivalent Econet SWI calls when the target station is accessible via raw Econet. This results directly in an Econet four-way handshake exchange. *Note:* MNS will *always* use raw Econet to access stations on the same Econet, for efficiency reasons.

D.2 Data transfer over UDP/IP

The format of message frames assembled by the MNS module when it needs to transmit over UDP/IP is as followa:



MNS Frame Format

A two way handshake mechanism is used to control data delivery. Field *FRAME_TYPE* specifies either *data_frame* (1) or *data_frame_ack* (2). Each data_frame must be positively acknowledged before the next transmission can begin. If a receiving station cannot deliver the message, for example due to lack of buffer space, then it will discard the message and not return an acknowledgement. The transmitting station will timeout and retransmit, according to the values of the *Count (R6)* and *Delay (R7)* parameters to the initiating user SWI calls. Lack of acknowledgemen after the specified number of retransmission will result in a *Status_NotListening (3)* status return value at the MNS SWI interface. The generic error value *Status_NetError (2)* will be returned on all other error conditions detected by the MNS software.

Field *FLAG_BYTE* contains the *flag byte (R0)* parameter to the user SWI call in a data_frame message.

Field *USER_DATA* is variable length in data_frame messages, absent in data frame_ack messages.

Fiekd *SEQUENCE_NUMBER* enables a receiving station to check that a received frame is not a retransmission of a previously acknowledged frame, which might occur if the earlier acknowledgement was lost in transit.

The *port number (R1)* SWI parameter maps directly into a UDP port number as an offset from the MNS base port number &8000. This means that MNS data transfer operations use UDP port numbers &8001 - &80FF, with the *Address Transform Protocol* using UDP port &8100.

D.3 Immediate operations

An immediate operation will be carried out via the following messages, which follow the basic format for MNS frames.

Field *FRAME_TYPE* specifies either *immediate_op* (3) or *immediate_op_reply* (4).

Field *FLAG_BYTE* specifies the operation type.

Field *SEQUENCE_NUMBER* protects against duplicated transmissions, as for data frames.

Field *USER_DATA* contains the immediate op data field.

The UDP port number used for immediate operations is &8000.