BIANCA/BRICK

Software Reference

Version 2.3
Document # 71040A

July 1999

NOTE

The information in this manual is subject to change without notice.

This manual provides a description of the system software for the BinTec BIANCA/BRICK family of ISDN multiprotocol routers.

While every effort has been made to ensure the accuracy of all information in this document, BinTec Communications AG assumes no liability to any party for any loss or damage caused by errors or omissions or by statements of any kind in this document.

The information in this manual is subject to change without notice. For additions or changes to this document please refer to the most recent version of this document and/or separate Release Notes which are available at BinTec's World Wide Web server at:

http://www.bintec.de

BinTec Communications AG July 1999

www.bintec.de

BIANCA/BRICK

Software Reference
Version 2.3

Contents

Configuring the BRICK as an IP Router

1

Chapter One

INTRODUCTION

What's Covered?

- Purpose of this document
- How to get the latest software and documentation
- How this document is organized
  - Sections
  - Document Navigation
- What's included in this document
- Conventions used in this guide

Purpose of this document

The *BIANCA/BRICK Software Reference* is provided in electronic form (Adobe's PDF (Portable Document Format)) as an addition to your printed documentation. This document describes the system software used on the BIANCA/BRICK family of routers: BRICK-XS, BRICK-XM, BRICK-XL, BRICK-XMP, BinGO!, BinGO Plus, and V!CAS. Note however that some information contained in this document is specific to certain hardware that may not be present on all products.

This document explains the BRICK system software in greater detail than covered in your printed documentation. The configuration examples contained throughout this manual are based on the BRICK's SNMP shell-interface. For a shorter description of configuring a particular BRICK feature please refer to your printed (or the online version) *User's Guide* or the online *Extended Feature Reference* from the *Companion CD*.

How to get the latest software and documentation

BinTec provides the most current versions of Software and Documentation via the World Wide Web (WWW). Our WWW server can be reached at:

http://www.bintec.de

Among other information you will find the most recent versions of:

- User Documentation for BIANCA/BRICK software/hardware.
- System Software images for your BIANCA/BRICK product.
- Release Notes for information about new features implemented for your BRICK or BinGO! system.
- Windows Software and UNIX Tools applications.
- Additional information (such as FAQs, White Papers, or other Product Information) that may be useful when working with your new BinTec router.

How this document is organized

Sections

Section 1 (chapters 1 - 5) contains information of general interest for individuals administrating the BIANCA/BRICK.

Section 2 (chapter 6 - 12) consists of detailed information relating to specific subsystems of the BIANCA/BRICK. Separate chapters are provided for IP Routing, IPX Routing, X.25 Routing, etc.

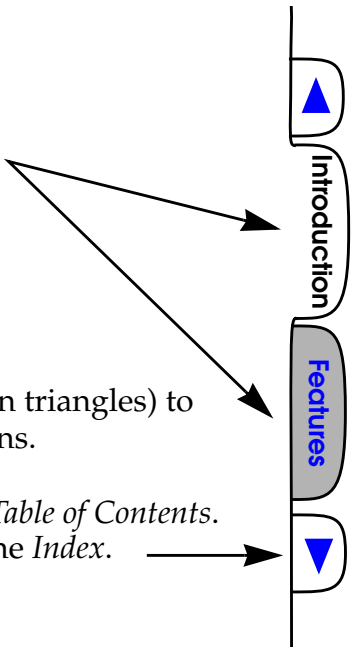
Section 3 (appendices A - E, and Glossary) contains supplemental reference material that may be useful when working with BIANCA/BRICK.

Document Navigation

In addition to the navigational aids provided by your PDF viewer application we have provided hypertext links (or hot-links) so you can quickly jump to other locations or supplemental material in your documentation.

Hyper Tabs

Use the chapter-tabs located along the right edge of the page so to jump to the beginning of a selected chapter within the current section.



Use the section-tabs, (Up and Down triangles) to jump to the previous or next sections.

Note:

In Section 1, Up takes you to the *Table of Contents*.

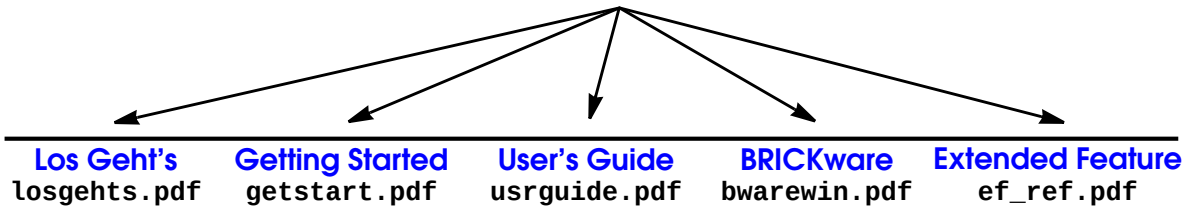
In Section 3, Down takes you to the *Index*.

Other Documents

Hot -links to other documents accompanying your BIANCA/BRICK product are located along the bottom of the page.

Note: These links are encoded relative to the current document.

If you're not viewing this document from the *Companion CD* (perhaps you've retrieved the newest version of this document via the WWW), the PDF files shown below must be located in the same directory (or via an appropriate UNIX file system link) as this document (and be named exactly as shown) for the links to work properly.



Los Geht's	Getting Started	User's Guide	BRICKware	Extended Feature
`losgehts.pdf`	`getstart.pdf`	`usrguide.pdf`	`bwarewin.pdf`	`ef_ref.pdf`

What's included in this document

Section One

Chapter 1 Introduction is this chapter.

Chapter 2 BRICK Features gives you a brief overview of some of the important features provided by your BIANCA/BRICK. Configuring and using these features is described in greater detail in the remaining chapters of this book.

Chapter 3 The SNMP shell describes the BRICK's SNMP shell, or command line interface. This chapter will help you understand some of the fundamental concepts used on the BRICK and become familiar with manipulating SNMP tables and variables from the command line.

Chapter 4 ISDN Connections on the BRICK describes how the BRICK handles ISDN connections and also includes some basic background information for those not familiar with ISDN networks.

Chapter 5 System Administration on the BRICK deals with administering the BRICK from the SNMP shell as well as concepts that apply to system administration in general.

Section Two

Chapter 6 Configuring the BRICK as a Bridge describes using your BRICK as a bridge.

Chapter 7 Configuring the BRICK as an IP Router describes using your BRICK as an IP router. Considering the BRICK is first and foremost an IP router, this chapter is by far the biggest chapter.

Chapter 8 Configuring the BRICK as an IPX Router describes using your BRICK as an IPX router to connect local and remote sites/LANs via Novell's IPX (Internet Packet Exchange) protocol.

Chapter 9 Using the BRICK as a CAPI Server covers the CAPI subsystem on the BRICK, and describes how you can use the BRICK to provide CAPI services (access to ISDN lines) to remote hosts on the LAN.

Chapter 10 Telephony Services on The BRICK covers the TAPI subsystem on the BRICK. This chapter describes how you can use the BRICK as a TAPI server to provide access to telephony services (analog lines) to remote hosts on the LAN running TAPI applications.

Appendix A CAPI Information Values contains CAPI 1.1 and CAPI 2.0 info values and their appropriate error-codes. This information is included for administrators that may need to debug CAPI applications using the BRICK as a CAPI server.

Appendix B Ethernet Framing describes the various frame formats the BRICK uses when sending data over LAN interfaces.

Appendix C ISDN Error Codes contains error codes used in Euro-ISDN and national ISDN (1TR6) in Germany, as well as some local error codes used on the BRICK. These will be useful for administrators dealing with problems involving ISDN signalling problems.

Appendix E Syslog Messages describes some of the system logging messages generated on the BRICK during special system events.

The Glossary of Networking Terms contains a brief listing of some of the more common acronyms and terms used throughout your user documentation and the networking field.

Conventions used in this guide

To help you locate and interpret information easily, the following visual clues and typographic conventions are used throughout this manual.

Visual Clues	
	This symbol is used to point out references to other helpful information such as man pages, other documents or chapters.
	This symbol is used to point out special information regarding safety concerns, possible error conditions, or hard-/software limitations.

Text displayed on a shaded background represents an example SNMP shell session, i.e., commands entered at the shell prompt and the output written to the screen.

```
mybrick:ipRouteTable > ping 10.1.1.5
64 bytes from 10.1.1.5 icmp_seq=0. time=1. ms
----10.1.1.5 PING Statistics----
1 packets transmitted, 1packets received, 0% packet loss
round-trip (ms)  min/avg/max = 1/1/1
mybrick:ipRouteTable >
```

Typographic Conventions
`Bold constant width font` is used within paragraph text to represent characters you enter on the command line.
Bold italic text represents system variables or table names
Blue underlined text is used to represent clickable hypertext references (or hot links) to other documents or sections.
Text enclosed in a box like this; SYSTEM represents a reference to a menu or submenu within Setup Tool.

2

Chapter Two

BRICK FEATURES

As a multiprotocol ISDN router, the BRICK supports too many networking protocols and ISDN features to cover in detail in a single chapter. The configuration of many of these features are tucked away in the BRICK's systems tables and can only be explained in light of their usage. This chapter gives you an overview of the major features found on the BRICK and references other locations within this document where they are explained in more detail.

What's Covered?

- ISDN Features
  - ISDN Protocol Support
  - V.110 Support
  - ISDN Callback Support
- IP Features
  - DHCP Server
  - DNS and WINS (NBNS) Negotiation over PPP
  - Dynamic IP Address Assignment
  - Extended IP Routing
  - IP Session Accounting
  - Network Address Translation
  - Proxy ARP
  - RIP Support
  - OSPF
- Security Features
  - Web based Monitoring
  - RADIUS support
  - IP Access Lists
  - Bridge Filtering
  - ISDN Call Screening

ISDN Features

ISDN Protocol Support

The BRICK supports the following ISDN protocols:

- Euro ISDN (Europe)
- National ISDN 1 (USA)
- National ISDN 2 (USA)
- NTT INS64 ISDN (Japan)
- 1TR6 National ISDN in Germany
- Northern Telecom DMS-100 (USA)
- AT&T 5ESS Custom ISDN (USA)

V.110 Support

V.110 is a ITU-T standard that defines the communications procedures to use when a communications device can't match the data rates offered by an ISDN and is aptly called bit rate adaption. Basically the transmitter and receiver have to agree to add additional bits during transmission to adjust the data rate to a mutually compatible rate. Asynchronous bit rate adaptation is often used in communication with terminal adapters and for connecting to GSM networks from the ISDN.

The BRICK supports bit rate adaption according to the V.110 standard for both incoming and outgoing calls. The type of bit rate adaption can be configured separately for each dialup PPP partner in the ***biboPPPTable***. This is explained in Chapter 4, ISDN Connections on the BRICK.

ISDN Callback Support

The BRICK supports ISDN Callback in both directions. Also an important security feature, callback can be configured on a per-partner basis to:

enabled	Here, the BRICK accepts an initial call from a specified partner. Upon succesful identification, the BRICK immediately closes the connection and returns the call.
expected	When callback is expected, the BRICK is the initiating party. The BRICK calls the specified partner, closes the connection, and waits (expects) the partner to return the call.

Configuring callback is covered in Chapter 7, Configuring the BRICK as an IP Router.

DHCP Server

The BRICK can be used as a DHCP (Dynamic Host Configuration Protocol) Server to manage networking resources for a number of local or remote DHCP clients. This is an efficient way of administering limited IP address resources. The BRICK supports DNS and WINS Relay.

Clients such as Windows 95 and Windows NT hosts can be configured to request networking resources from a DHCP server and to adjust their configurations appropriately. Configuring the BRICK as a DHCP server is covered in Chapter 7, Configuring the BRICK as an IP Router.

DNS and WINS (NBNS) Negotiation over PPP

The BRICK supports DNS and WINS Negotiation over PPP as specified in RFC 1877. This means that the BRICK is able to negotiate and configure its primary and secondary domain name servers and its primary and secondary NetBios name servers at connection time with compliant hosts.

DNS/WINS Negotiation over PPP can be configured separately for each PPP partner in the ***biboPPPTable***; this is covered in Chapter 7, Configuring the BRICK as an IP Router.

Dynamic IP Address Assignment

Dynamic IP address assignment in both client and server modes.

Client Mode	In client mode the BRICK is configured to accept it's own IP address after establishing an IP connection. This is useful for sites using low to mid-range BRICKs to connect to Internet Service Providers.
Server Mode	In server mode the BRICK assigns an available IP addresses from a preconfigured IP address pool. This is useful for any site using a BRICK product as a remote access point to a central LAN.

Dynamic IP address assignment can be configured for each partner separately in the ***biboPPPtable***. This is covered in Chapter 7, Configuring the BRICK as an IP Router.

Extended IP Routing

Most routers base IP routing decisions solely on an IP packet's destination address. With Extended IP Routing on the BRICK, routing decisions can be made based on additional information contained in the data packet. This gives you a much finer control over routing decisions and allows you to make routing decisions based on the contents of the IP packet:

- Type of Service (TOS field in ethernet frame)
- Source IP Address
- TCP Source Port
- TCP Destination Port

Routing decisions can also be based on BRICK interfaces:

- Source Interface
- State of the Destination Interface

The main advantage of extended IP routing is that traffic can be selectively routed over different transport mediums based on your site's needs. Some users require greater bandwidth for bulk data transmissions while others need shorter bursts for interactive sessions. Extended IP routing allows you to take advantage of different technologies (ISDN dialup, leased lines, X.25, and/or X.31 links) based on your site's specific needs.

Configuring Extended IP Routing is covered in Chapter 7, Configuring the BRICK as an IP Router.

IP Session Accounting

As an advanced IP feature, IP Session Accounting lets you generate BRICK accounting records for each TCP, UDP, or ICMP session routed over the BRICK. Accounting records contain information such as protocol usage, source and destination addresses, transfer activity, and the date, time, and duration of the IP session. By default, accounting records are written to the BRICK's system logging table but can also be forwarded to remote log hosts on the LAN for later processing. (See in Chapter 5).

Session accounting can be configured on a per-interface basis in the ***ipSessionTable***. This is covered in IP Session Accounting in Chapter 7.

Network Address Translation

With Network Address Translation, or NAT, the BRICK is able to hide a complete LAN behind a single IP address. This means that no matter how many users are connected to the LAN, only one official IP address is required to connect the complete LAN to the Internet. This address can also be a static address or dynamically assigned by an ISP at connection time.

NAT is accomplished by manipulating all incoming/outgoing IP packets to reflect different source and destination addresses. The translation process remains invisible to the connected networks. Hosts on the LAN continue to use standard IP addresses, however they are no longer accessible from hosts external to the LAN.

NAT is most useful where:

- Security is an issue. (controlling access to a limited number of hosts)
- The number of available IP addresses is limited.
- Monitoring of outgoing connections is desired.

NAT is an advanced IP feature. Configuring Network Address Translation is covered in Chapter 7.

Proxy ARP

Proxy ARP is supported on the BRICK for dial-in hosts that aren't connected directly to the LAN. With Proxy ARP the BRICK answers ARP requests for such hosts. To the local hosts the dial-in host appears to be on the LAN segment. Note that ARP (Address Resolution Protocol) is a standard method used to map IP addresses to physical MAC address.

Proxy ARP on the BRICK is straightforward; the respective interface is enabled in the ***ipExtIfTable***, the BRICK adjusts its routing tables automatically. See Proxy ARP in Chapter 7 for more information on configuring Proxy ARP.

RIP Support

The Routing Information Protocol is commonly used in IP networks to propagate routing information among routers. The BRICK supports version 1 and version 2 of RIP. Selected BRICK interfaces can be configured to independently send (and/or receive) version 1, version 2, both, or no RIP packets.

See the Advanced IP Features section in Chapter 7 for information on configuring RIP Options.

OSPF

The BRICK supports OSPF (Open Shortest Path First) and has been implemented according to the Internet standsards defined in RFCs 1583 (OSPF Version 2), 1793 (OSPF over Demand Circuits), and 1850 (OSPF Version 2 Management Information Base).

Special OSPF features such as MD5 authentification, importing of routing information via external protocols, and propagation of system-wide default routes is also supported.

For background info on the OSPF protocol refer to OSPF in Chapter 7. Configuring OSPF on the BRICK is explained in detail in the section Routing with OSPF.

Security Features

Web based Monitoring

A HTTP server is included on the BRICK and provides SNMP community password protected access to all system tables and variables via a TCP connection (port 80 by default). This means that the BRICK can be monitored via any WWW browser[1].

A built-in status page provides a quick overview of current operating state and hypertext links to all system information. A CGI program is also included and allows you to monitor selected system variables. Simply point a compatible web browser at the BRICK's status page as follows.

1. Browsers must support the HTML 2.0 standard and HTML tables (RFC 1942).

http://<*BRICK's System Name*>**<:**<*HTTP Port Number*>

More information on web based access to the BRICK is covered in Chapter 5, System Administration on the BRICK.

RADIUS support

RADIUS (Remote Authentication Dial In User Service), is an emerging client - server security system (initially developed by Livingston Enterprises, Inc.) to control access to network resources. The RADIUS server manages a database of user authentication data.

The BRICK can be configured to operate as a RADIUS client that consults the RADIUS server at connection time for specified dial-in partners. Partner specific connection parameters can be centrally managed in this way. This allows sites already using the RADIUS systems to centrally manage network resources, to easily integrate the BRICK into their existing network management system.

RADIUS support is configured using the ***biboAdmRadiusServer*** variable and the ***biboPPPTable***. This is covered in Chapter 7, Configuring the BRICK as an IP Router.

IP Access Lists

IP Access Lists provide you with the ability to fine tune access restrictions to and from connected IP networks. Access lists define the types of IP traffic that the BRICK should accept or deny (i.e., packets are either routed or are discarded). Access decisions are based on information contained in the IP packet such as:

- Source and/or Destination IP Address
- Source IP port (port ranges are supported)
- Destination IP port (port ranges are supported)

Sites using the BRICK to connect a LAN to the Internet for example might want to deny all incoming FTP requests, or outgoing telnet sessions from selected LAN hosts. Access Lists provide a powerful tool in controlling access to network resources. Refer to your User's Guide for more information.

Bridge Filtering

Bridge Filtering, sometimes called packet filtering, can be used to control the type and amount of traffic that is bridged over local interfaces. This is an important feature most useful when bridging over WAN links such as ISDN.

Bridge filtering is relevant for sites requiring bridging where:

- Minimizing ISDN costs is a concern.
- Greater control of bridging traffic is desired.

See the section Bridge Filtering in Chapter 5, Configuring the BRICK as a Bridge for detailed information.

ISDN Call Screening

The BRICK supports the call screening service provided by the ISDN and uses this service as an additional security measure to check the authenticity of incoming ISDN connections.

Call screening is mainly used in screening incoming PPP connections but can also be used to ensure access to the BRICK's isdnlogin service is secure. Refer to Chapter 4, ISDN Connections on the BRICK, for information on using the ISDN screening mechanism.

3

Chapter Three

THE SNMP SHELL

What's Covered?

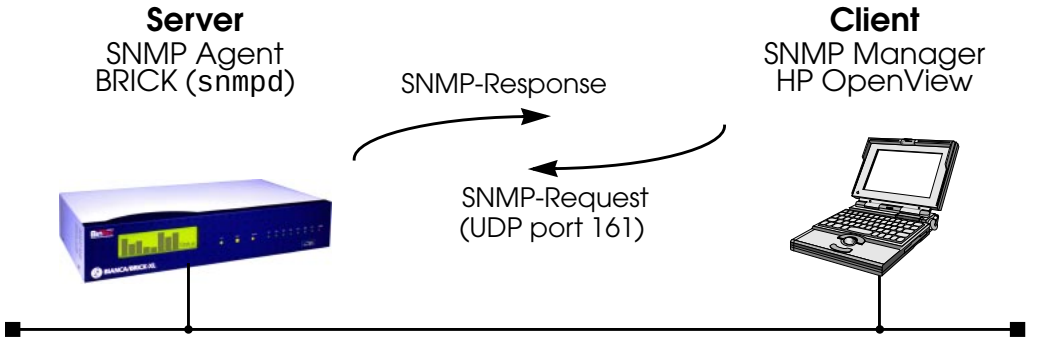
- SNMP Explained
  - Overview
  - The MIB
- SNMP Shell Overview
  - The Shell Prompt
  - Command Line Editing
  - Object Types
    - Integer Values
    - Enumerated Types
  - Shell Commands
  - External Commands
- BRICK System Tables
  - Short vs. Long Names
  - Creating Table Entries
  - Deleting Table Entries
  - Editing Table Entries
- BRICK Interfaces
  - Special Interfaces
  - Hardware Interfaces
  - Software Interfaces
- BRICK Configuration Files
  - Managing FLASH files
  - Transferring Files with TFTP
  - Transferring Files with XMODEM via Serial Port
  - Rebooting the System

SNMP Explained

Overview

SNMP (Simple Network Management Protocol), the successor to SGMP (Simple Gateway Monitoring Protocol), is used to manage network devices (workstation, terminal server, printer, bridge, hub). SNMP is an Application Layer protocol and uses the underlying UDP (User Datagram Protocol) as its transport medium; SNMP defines the rules of communication between an SNMP Manager and SNMP Agent allowing a network administrator to "watch" and/or control individual devices by viewing/changing operational settings stored on the managed device.

SNMP-based network management is a Client-Server system; however, the terms *Manager* and *Agent* are misleading in this context.



SNMP can be seen as a simple asynchronous **request–response** protocol. Messages are passed via UDP (normally port 161) and are binary in format. The SNMP Manager requests information from a specific device and an SNMP Agent (running on the device) authenticates the requester and responds with the requested information.

As mentioned above, different types of network devices may be managed via SNMP. Inherently, such systems have very different types of operational settings (comparing say a router to a printer). SNMP is not concerned with the contents of the messages being sent but with the methods used to obtain and change the settings on the remote devices. This is why SNMP is referred to as a simple protocol; because all network management functions basically boil down to a few basic operations. Operations available to the manager and agent processes are as follows.

Operations available to the SNMP Manager:

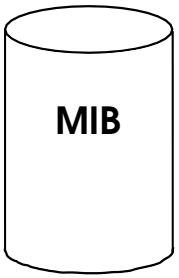
get-request	Requests the value associated with a specific variable.
get-next-request	Requests the next value associated with a variable that comprises a list of elements.
set-request	Sets or changes the value of a specific variable.

Operations available to the SNMP Agent:

get-response	Returns the value of a variable associated with a previous get-request or get-next-request message.
trap	Reports the occurrence of a fault condition, or other important relevant information.

The MIB

The MIB (Management Information Base) defines objects, often called MIB objects, that can be managed (i.e., queried, changed, created) for a particular device via SNMP. Objects per se are simply templates that define characteristics about a particular device and would include such things as:



- **Object Names**—How can the object be identified?
- **Object Descriptions**—How does the object's assigned value relate to the overall operational state of the device?
- **Object Access**—Who is allowed to change the object's setting?
- **Object Types**—Can the object's value be changed?
- **Object Ranges**—What values can be assigned to the object?

Consider IP routing for a multiprotocol router such as the BRICK. An IP route consists of several variables including at a minimum: Destination IP Address, IP Netmask, IP Metric, and Router Interface. Each of these items would be defined separately in the MIB. An example is an IP route's Next Hop object; though commonly referred to as ***ipNextHop*** its complete name is: **.iso.org.dod.internet.management.mib2.ip.iproutetable.ipNextHop**.

And in numerical form: .1.3.6.1.2.1.4.21.1.7 (see MIB Structure).

Also, a router's routing tables consist of multiple entries; thus multiple instances of the same object type would exist on a running system. This is a fundamental concept and means that the router needs a mechanism to uniquely identify a specific instance of an object. The naming structure used by the MIB provides this mechanism by asso-

ciating a MIB object with a local number and is called **Instance Identification**. When managing an IP router via SNMP it is *instances* of objects that are being manipulated.

SNMP Managers

Intelligent SNMP managers can communicate effectively with devices when the structure of MIB objects are known to it. ASCII files containing descriptions of MIB objects supported by a device are normally provided by the device's manufacturer and can be imported (or compiled) into SNMP manager applications.

MIB Structure

MIB objects have a hierarchical naming structure that forces every object to be unique to all other objects. This hierarchy is similar to a tree structure and is managed by the IANA (Internet Assigned Numbers Authority). Each node in the tree relates to a document that defines objects below that point. In SNMP individual object names are called Object Identifiers, or OIDs.

The figure on the following page shows the tree hierarchy relating to MIB objects implemented on the BIANCA/BRICK.

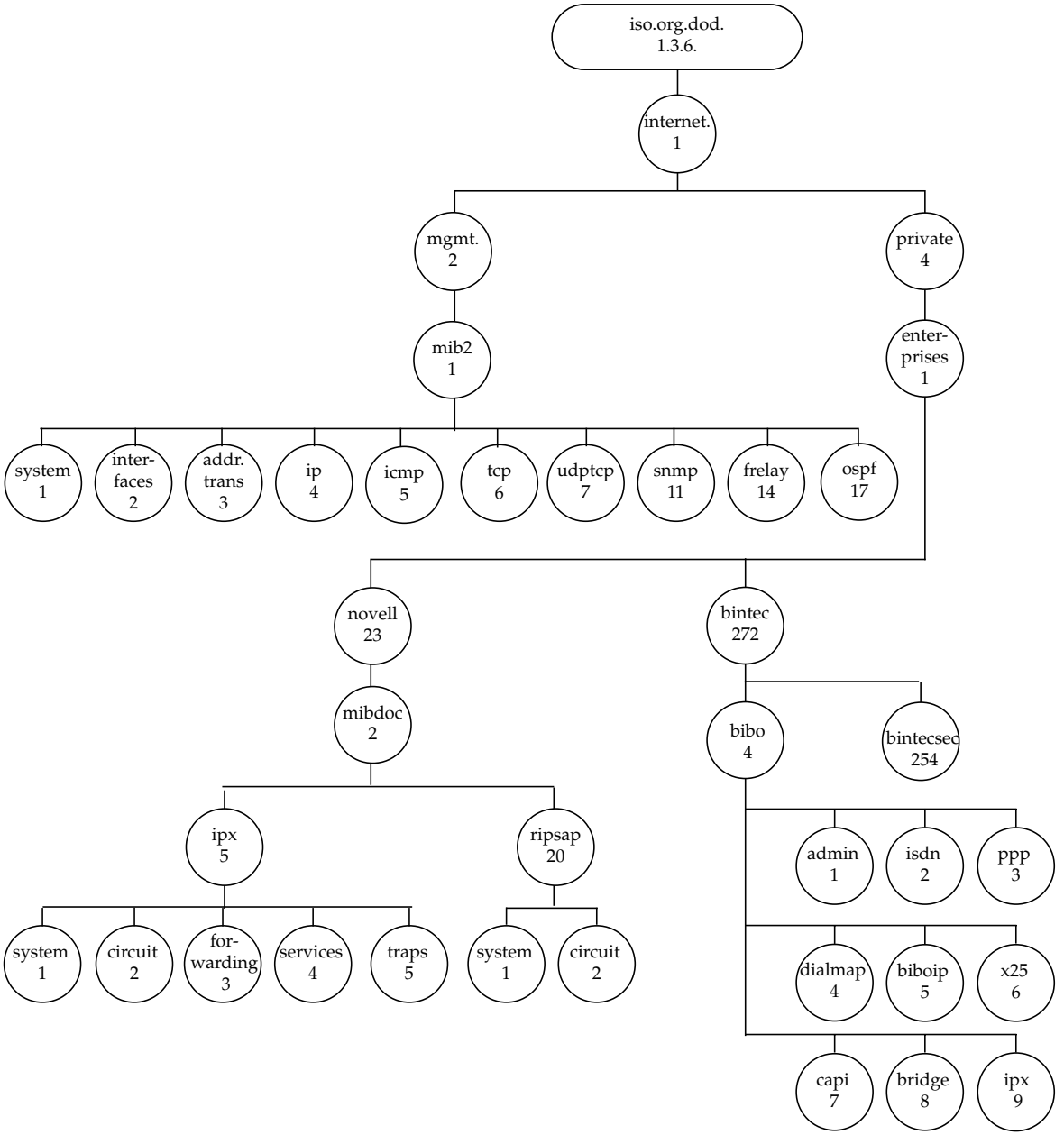
Note that OIDs can be referenced in two ways.

- Numerically
  Using the numbers assigned by the documents in the tree shown here a router's IP routing table is defined as object #21 of the ip module in the document that describes mib2.
  .1.3.6.1.2.1.4.21

- Textually
  Text names can also be used to identify objects. The router's IP route table would have the symbolic OID of:
  .iso.org.dod.internet.management.ip.iproutetable

Objects under the .iso.org.dod.internet.management tree are standard MIB objects defined by the ISO; enterprises (companies such as router manufacturers and protocol developers) may be assigned subtrees by IANA where their product specific objects can be defined. Since like devices provide similar services, and to provide interoperability between existing SNMP managers most devices support standard MIB objects defined by the ISO (International Organization for Standardization).

For internet routers MIB-2 (defined in RFC 1158) is the current standard and defines such objects as IP routing tables and various IP protocol settings. Multiprotocol routers that support IPX will also support Novell's enterprise MIB definitions.


iso.org.dod.
1.3.6.
internet.
1
mgmt.
2
private
4
mib2
1
enter-
prises
1
system
1
inter-
faces
2
addr.
trans
3
ip
4
icmp
5
tcp
6
udptcp
7
snmp
11
frelay
14
ospf
17
novell
23
bintec
272
mibdoc
2
bibo
4
bintecsec
254
ipx
5
ripsap
20
admin
1
isdn
2
ppp
3
system
1
circuit
2
for-
warding
3
services
4
traps
5
system
1
circuit
2
dialmap
4
biboip
5
x25
6
capi
7
bridge
8
ipx
9

SNMP Shell Overview

Along with other routing processes, the BRICK starts an SNMP Agent (see SNMP Explained) process when at boot time. You'll see these processes listed to the screen when the system is started (if a console is attached via the serial port). After all processes have been started, a login prompt is presented to the screen.

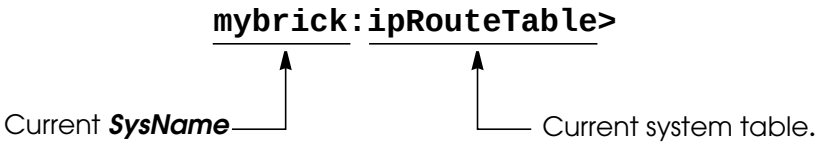
This login session is what we call the SNMP-shell.

The SNMP shell serves the same purpose as an SNMP Manager application. All of the BRICK's MIB objects can be managed from this shell. And because the shell is character based, the BRICK can be accessed remotely over any character-oriented connections such as:

- telnet sessions (from PCs or Workstations)
- HyperTerminal sessions (Windows 95 /Windows NT)
- isdnlogin sessions (isdnlogin command is provided)
- X.25 pad calls (minipad utility is provided)

The Shell Prompt

As shown below the shell prompt consists of two parts separated by a colon.



If the contents of the ***sysName*** object (system table) is not set, the first part of the prompt defaults to "brick".

Also, as you navigate among MIB objects the prompt will change to reflect the last system table displayed; similar to the "current working directory" variable used with many UNIX shells.

Knowing the current system table can be very useful when editing MIB objects because it allows you to use an object's short name instead of the complete object name; this is covered in the section Short vs. Long Names.

Command Line Editing

A command line editor is available from the SNMP shell. The command line editor allows you to edit commands on the command line before pressing the <Return> key let-

ting you adjust parameter settings or typing mistakes of previously entered commands. The up, down, right, and left arrow keys can be used as follows.

Key	Meaning
	Command History Moves you backwards through the list of commands entered during this shell session.
	Command History Moves you forwards through the list of commands entered during this shell session.
	Command Editing Moves the cursor backwards through the currently displayed command.
	Command Editing Moves the cursor forwards through the currently displayed command.

The command line editor is always in insert mode. Once the cursor is moved along the command line new characters typed in are inserted at the cursor's location. The <backspace> key can be used to erase characters.

Object Types

Each MIB object has a type associated with it that defines the types of values that it can be assigned. An object's type can be any of the following.

- Integer Value
- Character String
- IP Address
- Object Identifier
- Octet String
- Enumerated Value

For some objects the type of value that it may be assigned will be clear from the object's name. The ***Address*** variable in the ***biboPPPIpAssignTable*** is a good example, it accepts an IP address in dot format. You can determine an object's type from the SNMP shell by entering the object's name followed by a **?** (no space in between) and pressing <Return>. For example;

Objects that accept "integer values" can be set using one of four numbering systems as descibed below. Note however that some objects only accept numerical values in a

```
mybrick::system> ipRouteDest?
ipRouteDest: (readwrite) IP-address in dot-format (eg. 1.2.3.4)

mybrick::ipRouteTable> ipRouteInfo?
ipRouteInfo: (readwrite) object identifier in dot format (eg. .1.3.6.1)

mybrick::ipRouteTable> ipRouteType?
ipRouteType: (readwrite) other (1), delete (2), invalid (2), direct (3), indirect

mybrick:ipRouteTable> biboDialStkMask?
biboDialStkMask: (readwrite) binary integer (e.g. 0b1101)
```

specific numbering system such as "binary integers" as shown in the last example above.

Integer Values

When setting MIB objects that accept integer values the four numbering systems shown below may be used.

Numbering System	Prefix	Example Command	Resulting Decimal Value
Decimal	<none>	***ipDefaultTTL***=10	10
Octal	0	***ipDefaultTTL***=012	
Hexadeximal	0x	***ipDefaultTTL***=0xa	
Binary	0b	***ipDefaultTTL***=0b1010	

In most cases the decimal system is used; when using other numbering systems the above prefixes must be used to identify the appropriate numbering system.

Enumerated Types

Many MIB objects only accept values from a predefined list. These objects are said to be enumerated types. For example the ***Compression*** object in the ***biboPPPTable*** can be set to **`none`**, **`v42bis`**, or **`stac`**. No other values are acceptable.

The values for these objects are numbered starting at one with the first value being the objects default value. These numbers can also be used to set an object to the respective enumerated value. This means that the commands **`Compression=v42bis`** has the same effect as **`Compression=2`**

Shell Commands

The following commands are available from the SNMP shell.

Command	Usage	Meaning
Help	**?**	Lists all shell and external commands.
Community	**c** [<*community*>]	Sets/displays current SNMP community.
Group	**g** [<*groupnumber*> **|** <*groupname*> **|** ***]	Lists all groups or all tables within a group.
List	**l**	Lists all tables.
Priority	**p** [<*high*	*low*>]	Sets/displays current shell priority setting.
Columns	**u** [<*columns*>]	Sets/displays the number of columns used when displaying table output to screen.
Raw-Mode	**x**	Toggles shell's raw mode on and off.
Table-Mode	**y**	Toggles shell's table mode on and off.
Lines	**z** [<*lines*>]	Sets/displays the number of lines used when displaying table output to screen.
Exit	**exit**	Exits the current SNMP shell.

The help command (?)

Usage: **?**

The help command simply lists a summary of all available internal and external shell commands to the screen.

The Community Command (c)

Usage: **c** [<*communityname*>]

The community command sets or displays the current SNMP community name to use for SNMP command requests issued from the current shell.

Community names correspond to the password strings configured for the ***biboAdmAdminCommunity***, ***biboAdmReadCommunity***, and ***biboAdmWriteCommunity*** objects of ***bintecsec***.

For example, if:

1. The current value of ***biboAdmAdminCommunity*** = `bianca` AND
2. You are currently logged in as the `admin` user, AND
3. The community hasn't changed since logging in.

then your current community name (as displayed using **c**) is `bianca`.

Changing the community name during an SNMP shell session effectively changes read/write permission for MIB objects. This is similar to the UNIX su command except that no subshell is started.

If you log in as the `admin` user and change the value of ***biboAdmAdminCommunity***, the current community is automatically adjusted to the new value (with one exception as noted below).

NOTE

If you manually changed the community name any time during a shell session, the BRICK will no longer be able to automatically update the community name upon changes to ***biboAdmAdminCommunity.***

You will have to change the community name manually or log out and log in again using the new value.

The Group Command (g)

Usage: **g** [<*groupnumber*> | <*groupname*> | *****]

The group command lists BRICK subsystem groups, or all tables within a specific group to the screen. Options are used as follows:

<*groupnumber*> Specifies a group whose tables should be listed.

NOTE

The shell interprets integer values according to the format they are entered in. See Integer Values.

If you enter the command **g 08**, the shell interprets the leading 0 as identifying an octal and will report that the group doesn't exist. The command **g 010** would display tables in group eight; as would **g 0b1000**, **g 0xb**, and **g 8**.

<*groupname*> Specifies a group whose tables should be listed. Possible group names are those listed using the **g** command without parameters.

* Lists all tables without showing which group they belong to.

The List Command (l)

Usage: **l**

The list command lists all system tables to the screen. Table names are displayed in numerical order grouped by BRICK subsystem. A table's contents may be listed by entering the table name, or its number; i.e., entering **system** displays the same results as entering **1**.

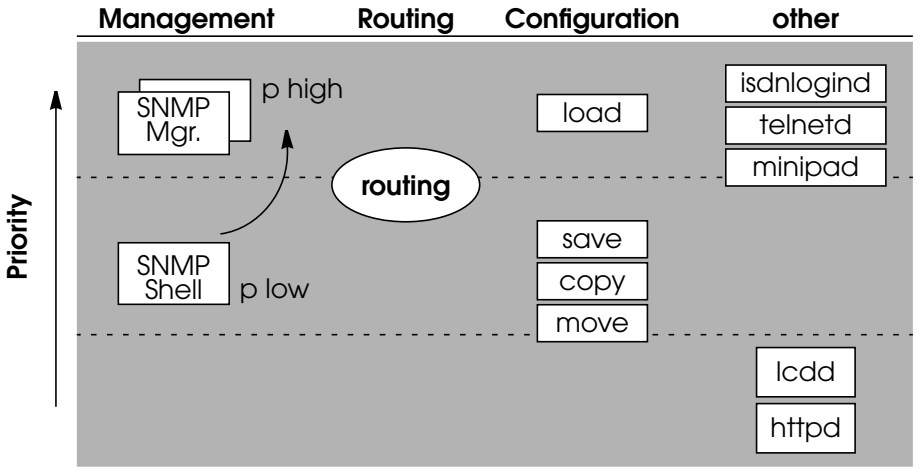
The Priority Command (p)

Usage: **p** [**high** | **low**]

The p (priority) command sets the priority (high or low) of the BRICK's SNMP shell with respect to other system processes.

The specified priority becomes effective for the current shell and all sub-processes started from this shell. When the p command is used without arguments the current shell priority is displayed.

By default, the SNMP shell has a lower priority than routing processes which means that an interactive configuration session (setup) will not affect performance on systems with many WAN partners.



Shell-priority and `cmd=save` commands:

If the shell's priority is set to **high** and a configuration is saved, the SNMP shell immediately returns you to the command prompt. This is in contrast to **low** status where the prompt is returned only after the save command is completed. The state of a `cmd=save` command can be verified by displaying the ***biboAdmConfigTable***.

Note:

This does not apply to SetupTool sessions.
SetupTool always waits for configuration management commands to complete before proceeding.

The Columns Command (u)

Usage: **u** [<*columns*>]

The columns command displays or sets the number of columns to use when writing to the screen; by default 79 columns. This is useful in Table Mode when using a non-standard sized terminal window (i.e., bigger than 80x24).

When using Setup Tool, the number of columns doesn't matter, since Setup Tool always displays output for a 79 column terminal window.

The Lines Command (z)

Usage: **z** [<*lines*>]

The lines command displays or sets the number of lines to use when writing to the screen; by default 24 lines. This is useful in Table-Mode when using a non standard sized terminal window (i.e., bigger than 80x24).

When in Table-Mode, the shell displays as many "complete" table entries as possible (<*lines required*> ≤ **z** <*lines*>) before prompting the user to continue with "**Press <RETURN> to continue or <q> to quit.**"

The Exit Command (exit)

Usage: **exit**

The exit command ends the current SNMP shell session and returns you to a new login prompt. Another way to exit the current shell is to enter Control-d, (holding down the Ctrl key and entering d).

The Raw-Mode Command (x)

Usage: **x**

The Raw-Mode command toggles Raw-Mode on and off. After entering the command, the shell reports which mode it is entering. By default Raw-Mode is off from the SNMP shell.

Raw-Mode means that MIB objects are dumped to the screen in the following format:

<MIB OID>.*<DB Key>*.*<Dynamic Number>* (*<Type>*) *<Value>*

MIB OID	The MIB Object's OID in numerical format.
DB Key	The DB Key identifies a specific instance of MIB OID and consists of the numerical values of all index variables for the system table.
Dynamic Number	A local number that the BRICK assigns locally.
Type	The MIB Object's type; integer, string, IP address, etc.
Value	The current value of this instance of MIB OID.

When Raw-Mode is on the current columns, lines, and Table-Mode settings are disregarded. Although the command prompt is not present when Raw-Mode is on, the command-line editor (cursor and backspace keys) can still be used. MIB objects can be set or queried using their numerical or symbolic names. Although this mode is not intended for interactive use it is possible to selectively display and set system variables. The difference between Raw-Mode on and off is shown below.

Raw-Mode on:

```
mybrick:system> x
rawmode on
ipNetToMediaTable
.1.3.6.1.2.1.4.22.1.1.1000.192.168.6.7.0      (Integer) 1000
.1.3.6.1.2.1.4.22.1.2.1000.192.168.6.7.0      (PhysAddress) 0:a0:f9:0:3:f2
.1.3.6.1.2.1.4.22.1.3.1000.192.168.6.7.0      (IpAdress) 192.168.6.7
.1.3.6.1.2.1.4.22.1.4.1000.192.168.6.7.0      (Integer) 4
.1.3.6.1.2.1.4.22.1.1.1000.192.54.54.2.1      (Integer) 1000
.1.3.6.1.2.1.4.22.1.2.1000.192.54.54.2.1      (PhysAddress) 0:0:f1:ab:f2:f3
.1.3.6.1.2.1.4.22.1.3.1000.192.54.54.2.1      (IpAdress) 192.54.54.2
.1.3.6.1.2.1.4.22.1.4.1000.192.54.54.2.1      (Integer) 4
```

Raw-Mode Off (default):

```
x
rawmode off
mybrick:system> ipNetToMediaTable

inx IfIndex(*rw)   PhysAddress(rw)     NetAddress(*rw)     Type(-rw)

 00 1000           0:a0:f9:0:3:f2      192.168.6.7         static

 01 1000           0:0:f1:ab:f2:f3     192.54.54.2         static

mybrick:ipNetToMediaTable>
```

The Table-Mode Command (y)

Usage: **y**

The Table-Mode command toggles Table-Mode on and off. After entering the command the shell reports which mode it is now entering. By default Table-Mode is on.

When Table-Mode is on the shell formats output using the current lines, and columns settings. Thus when a table's contents are displayed as many "complete" table entries are displayed as possible. The table's column names are displayed followed by the rows with each row separated by a blank line. If the table consists of more entries than can be displayed to the window (see the lines command), the user is prompted to continue.

Table-Mode On (default):

```
zeusbox:system> ipRouteTable

inx Dest(*rw)      IfIndex(rw)   Metric1(rw)      Metric2(rw)
    Metric3(rw)    Metric4(rw)   NextHop(rw)      Type(-rw)
    Proto(ro)      Age(rw)       Mask(rw)         Metric5(rw)
    Info(ro)

 00 192.168.12.0   1000          0                -1
    -1             -1            192.168.12.20    direct
    netmgmt        28674         255.255.255.128  -1
    .0.0

 01 192.168.6.0    1000           0               -1
    -1             -1            0.0.0.0          indirect
    netmgmt        28674         255.255.255.0    -1
    .0.0

 02 16.0.0.30      10001         0                -1
    -1             -1            16.0.0.30        direct
    netmgmt        28675         255.255.255.0    -1
    .0.0

Press <RETURN> to continue or <q> to quit.
```

Table-Mode Off:

```
15: ipRouteMetric2.192.168.4.128.15( rw):      -1
15: ipRouteMetric3.192.168.4.128.15( rw):      -1
15: ipRouteMetric4.192.168.4.128.15( rw): :    -1
15: ipRouteNextHop 192.168.4.128.15( rw):      192.168.4.128
15: ipRouteType.192.168.4.128.15(-rw):         indirect
15: ipRouteProto.192.168.4.128.15( ro):        rip
15: ipRouteAge.192.168.4.128.15( rw):          4
15: ipRouteMask.192.168.4.128.15( rw): :       255.255.255.128
15: ipRouteMetric5.192.168.4.128.15( rw) :     -1
15: ipRouteInfo.192.168.4.128.15( ro):         .:0.0
16: ipRouteDest.16.0.0.15.16( rw):             16.0.0.15
16: ipRouteIfIndex.16.0.0.15.16( rw):          1000
16: ipRouteMetric1.16.0.0.15.16( rw):          2
16: ipRouteMetric2.16.0.0.15.16( rw):          -1
16: ipRouteMetric3.16.0.0.15.16( rw):          -1
16: ipRouteMetric4.16.0.0.15.16( rw):          -1
16: ipRouteNextHop.16.0.0.15.16( rw):          16.0.0.15
16: ipRouteType.16.0.0.15.16(-rw):             indirect
16: ipRouteProto.16.0.0.15.16( ro):            rip
16: ipRouteAge.16.0.0.15.16( rw):              5
16: ipRouteMask.16.0.0.15.16( rw):             255.255.255.0
16: ipRouteMetric5.16.0.0.15.16( rw):          -1
16: ipRouteInfo.16.0.0.15.16( ro):             .0.0
zeusbox:ipRouteTable>
```

External Commands

As listed by the help (**?**) command the following external commands are also available from the SNMP shell.

- **ping**
- **ipxping**
- **setup**
- **ifstat**
- **debug**
- **ospfmon**
- **telnet**
- **minipad**
- **update**
- **netstat**
- **date**
- **traceroute**
- **isdnlogin**
- **halt**
- **ifconfig**
- **modem**

The `ping` Command

Usage: **ping** <*host*> [<*size*>]

The ping program can be used to test communication with another host. Ping sends ICMP echo_request packets of length *size* to *host*. Host is a required parameter which takes an IP address or a hostname. Size is optional and sets the length (in bytes) of the packets to use.

The ping command operates in continuous mode and keeps sending packets until the program is stopped by entering Control-C; that is, holding down the "Ctrl" key ("Strg" key on German keyboards) and pressing "C".

The `telnet` Command

Usage: **telnet** <*host*> [<*port*>]

The telnet program can be used to establish a terminal session with the host specified by the *host* parameter. The host's numerical IP address or hostname can be used. The optional *port* parameter specifies which TCP port to connect to on the host.

The `traceroute` Command

Usage: **traceroute** [**-m** <*maxhops*>] [**-p** <*port*>] [**-q** <*nqueries*>] [**-w** <*waittime*>] <*address*> [<*packetsize*>]

By using the Internet Protocol's "Time-To-Live" field, the traceroute program prints the route packets take to arrive at a network host. The only mandatory parameter is the destination *address* which may be the host's name or numerical IP address. Options are used as follows:

-m <*maxhops*>	The maximum number of hops probe packets may travel before reaching *host* (i.e., the value of each packet's TTL, Time-To-Live field).
-p <*port*>	The UDP port to use. By default port 33333 is used. Traceroute requires that *host* is not using ports between *port* and (*port* + *maxhops*-1). If needed this option should be used to select an unused port range.
-q <*nqueries*>	The number of queries to send, default is 3.
-w <*waittime*>	Seconds to wait for a response to a probe packet.
<*address*>	IP address (or hostname) of destination host.
<*packetsize*>	The size (in bytes) to use for each probe packet.

The `ipxping` Command

Usage: **`ipxping`** [**`-c`** <*count*>] [**`-d`** <*delay*>] [**`-s`**] <*net*> [<*node*>]

The ipxping command can be used to test communication between the BRICK and an IPX server and is comparable to IP's ping command. ipxping has one required argument, *net* which specifies the server's (or BRICK's) IPX Network Number. The optional arguments are used as follows:

-c <*count*>	Send exactly <*count*> packets. By default one packet is sent (that is, if both **-c** and **-s** are not used).
-d <*delay*>	The time (in seconds) to wait between packets. By default, 1 one second delay is used.
-s	Send 10000 packets.
<*node*>	Optional IPX node number. Should be used if the IPX host's Internal Network Number is not = 0:0:0:0:0:1.

Note:

Even if the BRICK's IPX network configuration is correct, the IPX server may not answer ipxping requests if it hasn't loaded its IPXRTR.NLM. It may be helpful to verify that the module is loaded if problems occur.

The `minipad` Command

Usage: **`minipad`** [**`-7`**] [**`-p`** <*pktsz*>] [**`-w`** <*winsz*>] [**`-c`** <*cug*>]
[**`-o`** <*outgocug*>] [**`-b`** <*bcug*>] <*x25address*>

The minipad program is a basic PAD (Packet Assembler/Disassembler) program that can be used to provide remote login services for remote X.25 hosts. Minipad is also useful for testing X.25 routes. To disable incoming X.25 connections to minipad, set ***x25LocalPadCall*** to **`dont_accept`**.

Minipad has one mandatory argument, *x25address*, which can be a standard X.121 address, when preceded by an "@", or an extended X.25 address. Data calls to closed user groups defined in the ***x25RouteTable*** and ***x25RewriteTable*** can be placed using the **`-c`**, **`-o`**, and **`-b`** options.

`-7`	Use 7-bit data bytes.
`-p` <*pktsz*>	The packet size to use.
`-w` <*winsz*>	The window size to use.
`-c` <*cug*>	Open connection with closed usergroup *cug*.
`-o` <*outgocug*>	Open connection with outgoing closed user group *outgocug*.
`-b` <*bcug*>	Open connection with bilateral closed user group *bcug*.
<*x25address*>	The remote host's X.25 address.

The `isdnlogin` Command

Usage: **`isdnlogin`** [**`-c`** <*stknumber*>] [**`-C`**] [**`-s`** <*service*>] [**`-a`** <*addinfo*>]
[**`-b`** <*bits*>] <*ISDN-number*> <*ISDN-servicename*>

The isdnlogin program enables you to start a remote login shell on the BRICK over ISDN. Using the *ISDN-number* and *ISDN-servicename* parameters, you select the ISDN partner to login to, and the ISDN service to use. Valid *ISDN* servicenames are shown below.

Through D-channel signalling, isdnlogin can also accept incoming calls from analog modem with V.110 bitrate adaption. Connections to V.110 stations can also be established with isdnlogin when the appropriate layer 1 protocol is supplied on the command line.

-c <*stknumber*>	The ISDN stack number to use.
-C	Attempt to use V42bis compression.
-s <*service*>	The 1TR6 service code to use for outgoing calls.
-a <*addinfo*>	The 1TR6 additional info code for outgoing calls.
-b <*bits*>	Use <*bits*>-bit data for transmission.
<*ISDN-number*>	The remote host's telephone number.
<*ISDN-servicename*>	The service names shown below are supported:

```
data          telephony    faxg3        faxg4
t_online      datex_j      btx          modem
56k           trans        dovb
v110_1200     v110_2400    v110_4800    v110_9600
v110_14400    v110_19200   v110_38400
```

The `setup` Command

Usage: **`setup`**

The setup command is used from the SNMP shell to start the Setup Tool program. Setup Tool provides a menu oriented interface to configuring the BRICK, its major features, and administering/monitoring its operational state. The User's Guide is completely Setup Tool based; please refer to it for information on using the Setup Tool program.

The `update` Command

Usage: **update** <*ipaddress*> <*filename*>

The update command can be used to upgrade the BRICK's internal system software using TFTP. The host at *ipaddress* can be a UNIX system or a PC as long as it's been configured as a TFTP server. For PCs, DIME Tools includes a TFTP Server application. For UNIX systems see the section Setting up a TFTP Server in Chapter 5. The *filename* specifies the image to load into flash ROM. This image must be present in the TFTP-root directory configured on the server.

The `halt` Command

Usage: **`halt`**

The halt command halts the system and reboots using the default boot configuration file. The halt command has the same effect as powering the system off and on; i.e. it immediately shuts down the BRICK. Therefore we recommend to better use `cmd=reboot`, because this way first running processes are completed, before the system is shut down (see Rebooting the System).

The `ifstat` Command

Usage: **`ifstat`** [**`-l`**] [**`-u`**] [<*ifcname*>]

The ifstat command displays status information for each of the system's interfaces, based on the contents of the ***ifTable***.

-l	Displays the full length of the interface descriptions (normally the description is limited to 12 characters).
-u	Only display information for interfaces in the **up** state.
‹*ifcname*›	Only display information for interfaces whose description starts with the given characters (e.g. **ifstat en1** displays information on the interfaces en1, en1-llc, and en1-snap).

Status information for the desired interfaces is displayed in eleven columns as shown below.

Column ***ifTable* Object**	Meaning
Index ***ifIndex***	The BRICK's interface number. Numbers > 10000 indicate software (or virtual) interfaces.
Descr ***ifDescr***	The interface's description as assigned to ***ifDescr***.

Column *ifTable* Object	Meaning
Ty ***ifType***	Interface type. Integer values correspond to the enumerated types of the ifType object. 6=`ethernet_csmacd`, 7=`iso88023_csmacd`, 23=`ppp`, 32=`frame_relay`.
Mtu ***ifMtu***	Maximum Transmission Unit for this interface; i.e., the largest network datagram that can be sent over this interface.
Speed ***ifSpeed***	The interface's estimated bandwidth in bits per second. For interfaces whose bandwidth doesn't change nominal bandwidth is reported.
St ***ifOperStatus***	The current operational status of the interface. May be: `up` (up), `dn` (down), `ts` (testing), `do` (dormant),or `bl` (blocked).
Ipkts ***ifInNUcastPkts ifInUcastPkts***	The number of packets received via this interface (the sum of ***ifInNUcastPkts*** and ***ifInUcastPkts*** objects) since ***sysUpTime***.
Ies ***ifInDiscards ifInErrors***	The number of incoming packets that couldn't be delivered due to errors (the sum of ***ifInDiscards*** and ***ifInErrors*** objects) since ***sysUpTime***.
Opkts ***ifOutNUcastPkts ifOutUcastPkts***	The number of packets requested via this interface (the sum of ***ifOutNUcastPkts*** and ***ifOutUcastPkts*** objects) since ***sysUpTime***.
Oes ***ifOutErrors ifOutDiscards***	The number of outgoing packets that couldn't be sent due to errors (the sum of ***ifOutDiscards*** and ***ifOutErrors*** objects) since ***sysUpTime***.
PhysAddress ***ifPhysAddress***	For hardware interfaces this is the physical (MAC) address. Software interfaces always display "point-to-point".

The `netstat` Command

Usage: **`netstat`** [**`-i`**] [**`-r`**] [**`-p`**]

The netstat command can be used to display a quick system status. Depending on which option is used, statistical information is retrieved from the ***biboDialTable***, ***ipxCircTable***, and ***ipRouteTable***. The three options are as follows:

`-i` Display status information for each interface. Output is displayed in 10 columns similar to the ifstat command. See the description of ifstat for information on each column.

`-p` Display information for each configured ISDN partner. Output is displayed in seven columns as follows:

Column	Meaning
`Index`	Interface number taken from ***biboPPPIfIndex***.
`Partnername`	The software interface's name as set in ***IfDescr***.
`Protocol`	The protocol configured for this interface as set in ***biboPPPEncapsulation***.
`State`	The current operational status as set in ***ifOperStatus***.
`Destination`	Associates IP address with this partner. The displayed address' type is specified in the Type field.
`Type`	Type of address specified in the Destination field, may be: `LOC` (local), `HOS` (host), `DEF` (default), or `NET` (network).
`Telno`	Lists telephone numbers configured for the partner (***biboDialNumber***). A number's direction (***biboDialDirection***) is indicated by a greater-than sign (>) for outgoing, a less-than sign (<) for incoming, or both (< >) for both in and outgoing.

`-r` Displays the current routing table entries. Output is displayed in seven columns as follows:

Column	Meaning
`Typ`	Type of address specified in the Destination field, may be: `LOC` (local), `HOS` (host), `DEF` (default), or `NET` (network).
`Destination`	The destination IP address for this route as set in the ***ipRouteDest*** object.
`Netmask`	The netmask for this route as set in ***ipRouteMask***.
`Gateway`	The IP address as set in ***ipRouteNextHop***.
`Met.`	The current operational status as set in ***ifOperStatus***.
`Interface`	The interface's name as set in ***ifDescr***
`Proto`	Identifies how this route was learned as stored in ***ipRouteProto*** (`local`=manually configured routes).

The `ifconfig` Command

Usage: **`ifconfig`** <*interface*> [**`destination`** <*destaddr*>] [<*address*>] [**`netmask`** <*mask*>] [**`up`** | **`down`** | **`dialup`**] [-] [**`metric`** <*n*>]

The ifconfig command can be used to assign an address to a network interface and/or to configure network interface parameters and change the respective routing table entries.

When only the required interface parameter is used, ifconfig displays the current settings for the interface.

Options (and their respective *ipRouteTable* entries) are used as follows:

<*interface*>	Interface name (***ifDescr***)
`destination` <*destaddr*>	Destination IP address of a host for adding host routes. (***ipRouteDest***, ***ipRouteMask***)
<*address*>	BRICK's IP Address for this interface. (***ipRouteNextHop***)
`netmask` <*mask*>	Netmask of interface (***ipRouteMask***)
`-`	Don't define own IP address (i.e. ***ipRouteNextHop*** = 0.0.0.0)
`metric` <*n*>	Sets route metric to *n* (***ipRouteMetric1***)

The debug Command

Usage: **debug** [**-t**] [**show** | **all** | [<*subs*> [<*subs*> ...]]]

The debug command can be used to selectively display debugging information originating from one or more of the various subsystems. Command line parameters are used as follows:

-t	Print a timestamp before each debugging message.
show	Show all possible subsystems that can be debugged. **ACCT ISDN INET X.25 IPX CAPI PPP BRIDGE CONFIG SNMP X.21 TOKEN ETHER RADIUS TAPI OSPF FR MODEM RIP**
all	Display debugging information for all subsystems.
<*subs*>	One or more subsystems separated by whitespace can be entered to display only debugging information from these subsystems. Current BRICK subsystems include:

The date Command

Usage: **date** [**-i**] [*YYMMDDHHMMSS*]

The date command is used to set or display the current time. All BRICK products have a software clock which stores the current time as retrieved from the host at ***biboAdmTimeServer***. The optional date-string sets the current date to the specified Year, Month, Day, Hour, Minute, and Second.

Note that the BRICK-XM and BRICK-XL also have a real-time clock (hardware).The **-i** option displays the date stored in the software clock and is therefore only available on the XM and XL

Product	Date Command:	
	date	**date** <*YYMMDDHHMMSS*>
V!CAS	Displays date currently stored in the *software* clock	Sets the *software* clock to <*YYMMDDHHMMSS*>
BRICK-XS		
BRICK-XM	Displays date currently stored in the *hardware* clock	Sets the hardware AND software clocks to <*YYMMDDHHMMSS*>
BRICK-XL		

The `modem` Command

Usage: **`modem`** [**`update`** <*TFTP host*> <*TFTP filename*> | **`status`**]

The modem command is used to update the system software of your CM-2XBRI module and FM-8MOD modem connector module, or to display the current operating status of all modems. Note that the FM-8MOD module is only available on the BRICK-XL. The command can be used as follows.

If the keyword **`update`** is used the following parameters are required.

<*TFTP host*>	The IP address of your TFTP server; i.e. the host where the modem software image can be retrieved.
<*TFTP file*>	The file name of the modem software image.

If you supplied the correct TFTP host and file name you will see some screen output concerning the loading and verifying of the image file. The update application will automatically detect all your modem connector modules and you will be queried to update each one individually.

If you reply with **`y`** the update will be performed. This will take approximately 60 seconds. After the modem update is complete you should reboot your BRICK immediately if you want to use the new modem software.

Note

To update the modem software image, a TFTP server (where your BRICK can retrieve the software image) must be configured (see Setting up a TFTP Server in Chapter 5 for additional information).

When the **`status`** keyword is used, the system displays the current status for each modem similar to the following.

No	State	OBytes	IBytes	LastMessage
00	IDLE	280	2704	CONNECT 115200/K56/LAPM/NONE/38000:TX/31200:RX
01	IDLE	278	2701	CONNECT 115200/V34/LAPM/V42BIS/33600:TX/33600:RX
02	IDLE	18481	22233	CONNECT 115200/K56/LAPM/NONE/40000:TX/31200:RX
03	CALLING	0	0	
04	CONNECTED	59635	64330	CONNECT 115200/V34/LAPM/NONE/33600:TX/33600:RX
05	CONNECTED	407	79	CONNECT 115200/K56/LAPM/V42BIS/36000:TX/31200:RX
06	CALLED	0	0	
07	IDLE	0	0	

The `ospfmon` Command

Usage: **`ospfmon db [rtr|net|sum|asbr|ext|stat]`** <*options*>

The ospfmon application can be used from the SNMP shell to display the contents of the BRICK's OSPF Link State Database. Note that only LSA header information is stored in the MIB system tables, this application can be used to dump the complete contents of the database. The various parameters can be used to selectively display specific types of database entries.

Only one of the six identiers can be used at time to display a cross section of the database.

`rtr`	Show all Router links.
`net`	Show all Network links.
`sum`	Show all Summary links.
`asbr`	Show all AS Border Router links.
`ext`	Show all External Links.
`stat`	Show OSPF database statistics.

Additional options may also be used to further identify more specific types of entries and include.

`area` <id>	Show database entries for area <*id*>.
`rtrid` <*id*>	Show entries generated by router ID <*id*>.
`lsid` <*id*>	Show database entry with link state ID <*id*>.

Example:

Router Links from the Link State Database for Area 0.0.0.0 (from BRICK-XL in this diagram) might look like this.

```
BRICK-XL:> ospfmon db rtr area 11.0.0.0
Area 11.0.0.0

    Router Link Age 920 Options 0x20 LsId 192.168.30.1
    RtrId 192.168.30.1Seq 0x80000002  Checksum 0xe72a Len 48
        options 0x2 links 2
        Stub Network id 12.0.0.2 data 255.255.255.255 metric 1562
        Stub Network id 12.0.0.3 data 255.255.255.255 metric 0
```

Note that the Link State ID (Lsid) of the database entry has different meanings based on the type of Link State Advertisement that is displayed. The table below shows the meanings for the five LSA types.

LSA Type:	Meaning of Link State ID:
Router Link	The **Router ID** of the router that generated the LSA.
Network Link	The **IP Address** of the DR on the destination network
Summary Link	The ***ipRouteDest*** of the propagated IP route.
ASBR Summary Link	The **Router ID** of the Autonomous System Border Router.
External Link	The ***ipRouteDest*** of the propagated IP route.

BRICK System Tables

When booting the BRICK loads its configuration file into memory. Normally the configuration file (named "boot") is loaded from flash memory. (A configuration file can also be loaded from a remote TFTP host at any time during operation.)

The BRICK's configuration file consists of system tables and variables whose format and structure are defined in The MIB. Upon loading this information is stored in memory (RAM) and can be seen as a sort of relational database whose current contents can be manipulated from the SNMP shell. Each table in the database consists of rows and columns where:

- Column headings represent individual MIB object type.
- Rows consist of instances of several MIB objects.

There are static tables only containing just one row (e.g. ***system***). Tables with multiple rows are numbered (inx) starting from 00., Thus each table entry, or row, refers to

an instance of several MIB objects, or variables. The ***ipNetToMediaTable*** (the current ARP cache) is shown below.

	ipNetToMediaTable			
inx	**IfIndex**(*rw)	**PhysAddress**(rw)	**NetAddress**(*rw)	**Type**(-rw)
00	1000	8:0:24:af:b2:3	192.168.6.140	static
01	1000	0:a0:f9:c7:4:4	192.168.6.12	static
02	2000	0:8:2:4b:4e:24	192.168.6.5	dynamic
03	2000	0:af:92:5a:1:2	192.168.6.37	dynamic

The characters (in parentheses) following each column name have special meanings for creating and deleting table entries. The *inx* number identifies a specific row and can be used when editing table entries.

*	Identifies **index** objects. Index objects define a unique database key that is required when creating new table entries.
-	Identifies the variable that contains the **delete** flag. These variables are used to delete a table entry.
ro	Identifies a variable as being **Read-Only**. These variables contain values that may not be changed.
rw	Identifies a variable as being **Read-Write**. Values for these variables can be changed.

Short vs. Long Names

When **Creating**, **Deleting**, or **Editing** BRICK system table entries, MIB variables are normally identified from the command line using their complete (or **Long Name**) name as defined in The MIB. Long Names for the MIB objects defined in the ***ipNetToMediaTable*** are:

```
ipNetToMediaIfIndex
ipNetToMediaPhysAddress
ipNetToMediaNetAddress
ipNetToMediaType
```

Note that objects contained in the system table currently displayed in The Shell Prompt are also accessible via their **Short Names**. This allows the shell prompt to op-

erate as a sort of *Current Working Directory*. Before changing (or creating) table entries from the SNMP shell, you'll probably want to display the table's contents first. As the table's contents are written to the screen the table's short names are displayed. The short names for MIB objects contained in the ***ipNetToMediaTable*** (shown on the previous page) are:

```
IfIndex
PhysAddress
NetAddress
Type
```

Note:

MIB objects are NOT case sensitive. Upper and lower case characters have been used above for added readability. System table entries can be manipulated using any combination of upper/lower case characters with either long or short names as explained above.

Creating Table Entries

Creating table entries is comparable to adding a new entry into the database that's currently stored in memory.[1] Since variables in the database are individual instances of MIB objects each variable must be identified by a unique key. Each table row contains a unique database key which consists of the values of all the index objects for that row. And because a table row can only contain one instance for each MIB object, the key identifies the instances of all variables for the row.

1.The reference to "memory" is here because changes to a table's contents are only saved to a writeable medium (flash ROM or a remote system's disk) upon explicit instruction.

In the ***ipNetToMediaTable*** shown above, the ***IfIndex*** and the ***NetAddress*** objects are index variables. The four instances of the ***PhysAddress*** object can be uniquely identified in the database their respective keys as follows:

PhysAddress Instance	DB Key (*IfIndex.NetAddress*)
8:0:24:af:b2:3	1000.192.168.6.140
0:a0:f9:c7:4:4	1000.192.168.6.12
0:8:2:4b:4e:24	2000.192.168.6.5
0:af:92:5a:1:2	2000.192.168.6.37

For index objects that are Enumerated Types the numeric value is always used. If the ***IfIndex*** object consisted of the values `ethernet (1)`, `token_ring (2)`, or `other (3)` the respective keys shown above might be: 1.192.168.6.140, 1.192.168.6.12, 2.192.168.6.5, and 2.192.168.6.37.

This complicated explanation simply means that in order to create a new table entry, a new database key has to be defined which involves setting all index variables within one command. Additional variables may also be set at the same time. Variables not defined when a row is created are assigned default values that may be changed later (see Editing Table Entries).

A new static ARP mapping entry (comparable to: `arp -s` on most operating systems) could be added to the ***ipNetToMediaTable*** shown above using the following command.

```
mybrick:system> ipNetToMediaIfIndex=1000 ipNetToMediaNetAddress=192.168.6.6
                ipNetToMediaPhysAddress=0:4:f1:a0:8:f3

mybrick:ipNetToMediaTable>
```

In this example, setting the ***ipNetToMediaPhysAddress*** object is not actually required for creating the table entry, however, it makes sense to associate the IP address with the hardware address within the same command.

Some system tables contain MIB objects for which only one instance is possible (the ***admin*** table for example; which among other things, contains the TCP port numbers the BRICK uses). Logically these tables (called static tables) can only contain one row.

NOTE

Some system tables are not intended to be changed manually (e.g., tables that contain ISDN call or IP session logging information) an only contain Read-Only variables. Although MIB objects are marked as index variables (with an *****) in these tables they're also marked **`ro`**; meaning that only the BRICK can update these tables.

Deleting Table Entries

To delete a table entry the variable (in the row that you want to delete) containing the delete flag must be set to `delete`. The delete flag is denoted by the (-) character in parentheses in the column name.

As mentioned above the database key identifies all instances of MIB objects on a specific table row. This key could be used to identify a specific instance of the delete object thereby deleting a complete table row. The format is <*MIB object*>**:**<*DB Key*>**=delete**. The first row in our ***ipNetToMediaTable*** could be deleted in this manner using the command:

ipNetToMediaType.1000.192.168.6.140=delete

So that you don't have to decipher database keys (which can sometimes be long and consist of multiple variables) the best way to remove a table entry is to append the table row number to the delete object (separated by a colon). The format is <*MIB Object*>**:**<*inx number*>**=delete**. The same row could be deleted from our ***ipNetToMediaTable*** using:

ipNetToMediaType:0=delete

or

ipNetToMediaTable:0=delete

The row numbers of each table are indicated by the ***inx*** number which is shown when displaying a table's contents to the screen.

Editing Table Entries

The contents of a specific instance of a MIB object, i.e., the contents of a specific table cell, can be changed. Both methods mentioned in Deleting Table Entries can be used.

Again the preferable (easier) method involves using the ***inx*** value to identify the table row.

We could change the hardware address associated with IP address 192.168.6.5 in our *ipNetToMediaTable* with either of the following commands.

ipNetToMediaPhysAddress.1000.192.168.6.5=0:0:f3:a0:3:f1

ipNetToMediaPhysAddress:0=0:0:f3:a0:3:f1

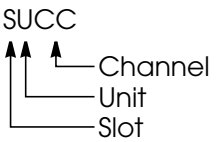
Of course variables can only be assigned values that are appropriate to the respective Object Types.

BRICK Interfaces

One of the key concepts used on the BRICK is the idea of interfaces; however, several different types of interfaces are used. These include the following which are described below.

- Special Interfaces
- Hardware Interfaces (i.e., the physical interface)
- Software Interfaces (also referred to as *virtual* interfaces)

The numeric value of the ***ifIndex*** variable, used in many BRICK system tables identifies a specific BRICK interface. The ***ifIndex*** is a five digit number (leading 0s are normally not shown) that identifies the interface's type and some of the special characteristics of the interface which are described in the following sections.

Type	Range	Comments
Special Interfaces	0	The REFUSE Interface
	1	The LOCAL Interface
	2	The IGNORE Interface
Hardware Interfaces (*Physical Interfaces*)	1000	
	...	
	9999	
Software Interfaces (*Virtual Interfaces*)	10000	Software interfaces sequentially ordered by category, see: (Software Interfaces)
	...	
	99999	

This shows the initial breakdown based on the interface types; software and hardware interfaces can be broken down further according to their specific characteristics. This is explained in the following sections.

Special Interfaces

Three special (destination) interfaces are available on the BRICK and are mainly useful when creating special routes for handling different situations depending on the characteristics of the interface.

These interfaces are always listed first in the ***ifTable*** (Interface Table) and have the following characteristics.

The REFUSE Interface (ifIndex = 0)

When packets are routed to the REFUSE interface (in the ***ipRouteTable*** and the ***ipExtRtTable***) the packet is discarded and an "ICMP Destination unreachable" message is transmitted to the sender; i.e., the host at the address identified in the Source IP Address field of the IP datagram (see the diagram of the Internet Layer in Chapter 6).

The LOCAL Interface (ifIndex = 1)

Packets routed to the LOCAL interface are given to an appropriate internal process on the BRICK such as the BRICK's minipad application.

The IGNORE Interface (ifIndex 2)

Packets routed (via the ***ipRouteTable*** and the ***ipExtRtTable***) to the IGNORE interface are discarded meaning that the packet is not forwarded. This destination interface is similar to the REFUSE interface with one exception. No status information is transmitted to the sender for packets routed to this interface.

Hardware Interfaces

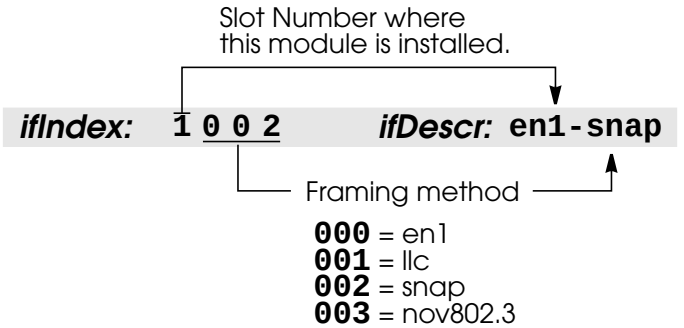
BRICK hardware modules are listed by SBus slot in the ***biboAdmBoardTable***. Both Feature Modules (FM) and Communications Modules (CM) are shown there. Communications modules that provide hardware interfaces routing capable of routing are listed in the ***ifTable*** and are identified by ***ifIndex*** values that are in the range:

ifIndex	**HW Interface**	***ifIndex***
1000 ≤	value	< 10000

These interfaces consist of Point-To-Multipoint interfaces (such as ethernet, and token ring interfaces), and Point-To-Point interfaces (such as ISDN S_0, ISDN S_{2M}, and X.21 interfaces).

Point-to-Multipoint

Point-to-multipoint interfaces (Ethernet and Token-Ring) are listed in the ***ifTable***. The value of the ***ifIndex*** and ***ifDescr*** objects are encoded as follows.



The ***ifIndex*** shown above identifies a point-to-multipoint interface installed in slot 1. On most systems this is the CM-BNCTP ethernet module but may also be a token ring module (CM-TR). The ***biboAdmBoardTable*** entries would verify this module. This hardware interface uses SNAP framing. For information on the frame formats used with point-to-multipoint interfaces refer to Appendix B).

Point-to-Point

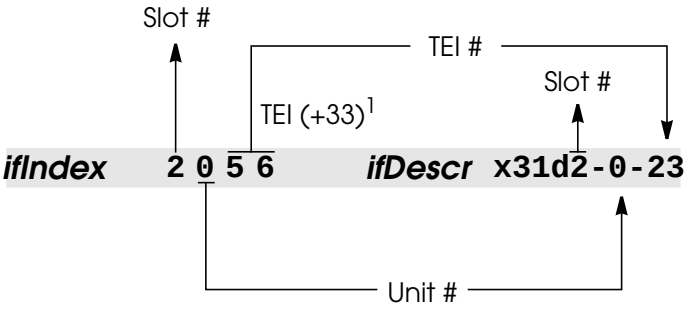
Point-to-point interfaces include various forms of X.21, X.25, and ISDN interfaces. These different types of point-to-point interfaces depend on the type of installed hardware, how the hardware is configured, and where (which SBus slot) the hardware is installed.

X.21 Interfaces

- X.21 interfaces are listed in the ***ifTable***. Only the slot digit is used and identifies the applicable slot for the CM-X21 module (i.e., 3000 for CM-X21 in slot 3). A corresponding entry (***x21IfIndex***) is also present in the ***x21IfTable***.

X.31 Interfaces

- X.31 (in the D-Channel) Interfaces
  X.31 interfaces are listed in the ***ifTable***. A corresponding entry is also found in the ***x25LinkPresetTable***. The ***ifIndex*** used for X.31 interfaces is encoded as follows.

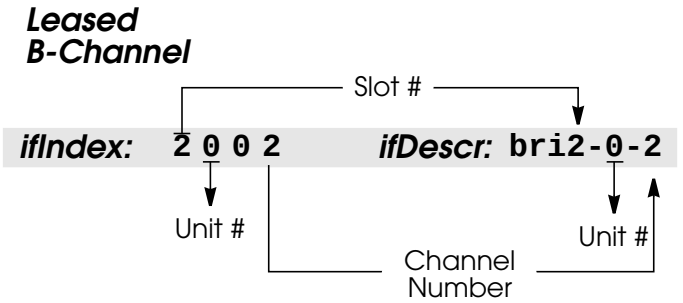


1.) 0 - 32 are reserved for ISDN leased B-Channel interfaces

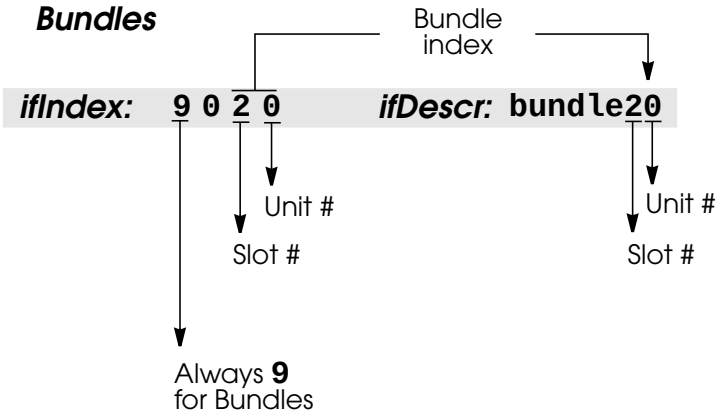
ISDN Interfaces

- **Dialup Interfaces**
  ISDN Dialup interfaces are not listed in the ifTable since they do not provide directly routeable interfaces; this is where the software interfaces are required.
- **ISDN Leased Line Interfaces**
  Leased line interfaces are listed in the ***ifTable*** since these interfaces identify a directly routable interface. Two types of leased line interfaces exist: interfaces

consisting of a single ISDN B-channel (S_0) and interfaces that consist of multiple ISDN B-Channels (S_0 or S_{2M}), called Bundles.
For Leased line interfaces consisting of a single B-Channel, the ***ifIndex*** and ***ifDescr*** objects are encoded as follows:



For bundle interfaces the ***ifIndex*** and ***ifDescr*** objects are encoded as follows:



Software Interfaces

Software interfaces are also referred to as *virtual* interfaces since they are mapped to one or more hardware interfaces (point-to-point or point-to-multipoint). The most common examples are dial-up ISDN partner interfaces. When setting up these software interfaces you may decide to associate one or more ISDN B-channels from one or

more ISDN ports (BRICK-XM and BRICK-XL only) that are used to accept calls from, or place calls to, the partner.

Software interfaces are listed in the ***ifTable*** and are identified by ***ifIndex*** values that are in the range:

ifIndex		**SW Interface**		***ifIndex***
10001	≤	value	<	29999

Software interfaces can be further distinguished as follows:

ifIndex	**SW Interface Type**
10001 ... 14999	Dial-Up ISDN Interfaces
15001 ... 15999	RADIUS Interfaces
18001 ... 19999	Frame Relay over ISDN Interfaces
20000 ... 29999	Multiprotocol over X.25 Interfaces

BRICK Configuration Files

BRICK configuration files are stored either locally on the BRICK or on a remote host (TFTP server). When booting this information is loaded into memory and becomes the BRICK's active configuration.

Configuration files stored locally on the BRICK are stored in FLASH PROM (programmable read-only memory) memory which we simply refer to as flash. The contents of flash can be seen as a directory whose contents are listed in the ***biboAdmConfigDirTable***. Configuration files stored in flash can be managed using the commands described in Managing FLASH files below.

Configuration files can be sent to or retrieved from remote hosts using the TFTP commands described in Transferring Files with TFTP.

The system can also be rebooted using the **`cmd=reboot`** command described in Rebooting the System.

Managing FLASH files

To help you manage different configuration files, the BRICK uses the ***biboAdmConfigTable***. This table contains the fields *Cmd*, *Object*, *Path*, *Pathnew*, *Host*, *State*, and *File*. This table is read by the configuration daemon, the **`configd`** process, which periodically:

1. Reads table entries.
2. Performs requested actions, using the respective field values as command parameters.
3. Updates the respective State field according to the State of the requested command.
4. Removes table entries once the respective action is performed.

An action is requested by assigning a value to the fields appropriate to the command, the **`configd`** process executes the requested actions.

The *State* field is updated intermittently while performing the action. The *State* field may be; *todo*, *running*, *done*, or *error*, depending on the status of the requested action. If the command resulted an error condition, you can find a detailed explanation of what caused the error by viewing the ***biboAdmSyslogTable***. Once the requested action is completed the results can be seen by viewing the ***biboAdmConfigDirTable***.

Configuration files can be stored on the BRICK (i.e., in flash) using the commands shown below. Although this information is presented in command syntax notation the

Tip When using third party SNMP managers configuration data can be managed by accessing the respective objects in the ***biboAdmConfigTable***.

actual commands simply involves assigning various parameter values to the contents of the ***biboAdmConfigTable***.

cmd=save [**path=**<*dirname*>] [**object=**<*tableobj*>]
cmd=load [**path=**<*dirname*>] [**object=**<*tableobj*>]
cmd=delete **path=**<*dirname*> [**object=**<*tableobj*>]
cmd=copy **path=**<*oldname*> **pathnew=**<*newname*>
cmd=move **path=**<*oldname*> **pathnew=**<*newname*>

Saving Configuration Files

The BRICK allows you to have multiple configuration files as long as there is enough room in flash to store them. To make sure that your configuration information (and any changes you have made while the system is running) is available after every system bootup, you must instruct the BRICK to write the configuration data. This is done by assigning the value “save” to the ***biboAdmConfigCmd*** field.

Usage:
cmd=save [**path=**<*dirname*>] [**object=**<*tableobj*>]

Optional arguments:

path Specifies the file in flash to write data to. The default value is “boot”. If *dirname* contains spaces, the name must be enclosed in double quotes.

object Specifies the object(s) to save. Either a specific table of information can be saved or a complete configuration. If no objects are specified, all tables are written to path.

The result of this command is that all Read-Write information is written to the file specified by path. If the optional parameters are not used, the complete configuration (all tables) is saved to the "boot." file.

For example,

cmd=save

would write all configuration information to flash as “boot”. If you want to save just the ***ipRouteTable*** in the file `test`, you could issue:

```
cmd=save path=test object=ipRouteTable
```

You can verify the actions have been completed by listing the entries in ***biboAdmConfigDirTable***. You should see a listing of each configuration file you saved. Each line shows you the name of the file (*Name*), the number of tables saved in the file (*Count*), and the contents of the file (*Contents*), i.e., “all” or a list of individual table numbers separated by colons.

Note:

The internal procedure of writing configuration files can take between 5 and 20 seconds. It is strongly recommended that during this time no additional changes be made to the configuration. Verify your current changes before making additional ones.

Loading Configuration Files

During system initialization, the default configuration file ("boot") is loaded into memory. This boot file may be loaded locally or via a remote system, using BootP. The working state of the BRICK is dependent upon on the configuration information in active memory. A new configuration file (or a single table) can be loaded into memory from flash while the system is running. This is done by assigning load to ***biboAdmConfigCmd***.

Usage:
`cmd=load` [**`path=`**<*dirname*>] [**`object=`**<*tableobj*>]

Optional arguments:

`path`	Specifies the file in flash to load data from. Default is “boot”. If *dirname* contains spaces, the name must be enclosed in double quotes.
`object`	Specifies the object(s) to load into memory. All tables can be loaded from a file or individual tables. If no objects are specified, all tables are loaded from path.

For example, to load a configuration from flash from the file “test”

```
cmd=load path=test
```

would be used; while the command

```
cmd=load path=test object=admin
```

would only load the ***Admin*** table from "test". After loading configuration information, changes take effect automatically since the information is loaded directly into memory.

Deleting Configuration Files

Deleting complete configuration files or specific tables within them is done by assigning "delete" to the *Cmd* field.

Usage:
cmd=delete path=<*dirname*> [**object=**<*tableobj*>]

Required arguments:
path Specifies the file in flash RAM to remove data from. Default is "boot". If *dirname* contains spaces, the name must be enclosed in double quotes.

Optional arguments:
object Specifies the objects to remove. If no objects are specified, all tables are removed from <*path*>.

For example, to delete all configuration information from flash file "test",

```
cmd=delete path=test
```

could be used; to delete only the ***ipRouteTable*** from the flash file "test" you could use

```
cmd=delete path=test object=ipRouteTable
```

Copying Configuration Files

To copy configuration files you can assign the value "copy" to the *biboAdmConfigCmd* object. When using "copy" the parameters differ slightly from their previously discussed usage.

Tip When deleting configuration files you may notice that the amount of available memory space shown in the ***biboAdmConfigDirTable*** is not adjusted. This is because flash can't be erased progressively; configuration files are only marked for deletion. When the flash becomes full, the system automatically reorganizes flash RAM, deleting previously marked data.

Note:  You can delete the contents of the flash RAM completely by assigning "/" to the path parameter. It is recommended however, that you save all configuration information to a remote host using TFTP and the cmd=put assignment before using this syntax.

Usage:
cmd=copy path=<*oldname*> **pathnew=**<*newname*>

Required arguments:

path Specifies the file in flash to copy data from. If *path* contains spaces, it must be enclosed in double quotes.

pathnew Specifies the new file in flash to write the data to. If *pathnew* contains spaces, it must be enclosed in double quotes.

This command is not capable of selecting individual tables from path; only complete files can be copied. Thus, the command:

cmd=copy path=boot pathnew=backup

copies the configuration file "boot" to the file "backup".

Moving Configuration Files

You can assign the value "move" to the *Cmd* field to rename configuration files. This has the same effect as issuing the cmd=copy and cmd=delete commands consecutively.

Usage:
cmd=move path=<*oldname*> **pathnew=**<*newname*>

Required arguments:

path Specifies the file in flash to remove.

pathnew Specifies the new file in flash to create.

The result of this operation is that the file <*oldname*> is renamed to <*newname*>. To rename the "boot" file to "oldboot" use the command:

```
cmd=move path=boot pathnew=oldboot
```

Transferring Files with TFTP

Using TFTP (Trivial File Transfer Protocol) you can transmit and retrieve configuration files to and from remote hosts on your network. This is made possible using three additional enumerated values for the ***biboAdmConfigCmd*** object: **put**, **get**, and **state**.

To exchange configuration files with remote hosts you must first set up a TFTP server on these hosts. Information on setting up a TFTP server on UNIX machines is provided in Chapter 5 in Setting up a TFTP Server. A TFTP Server application for PCs is included with *BRICKware for Windows*.

The commands used for exchanging configuration information among remote TFTP hosts are as follows.

cmd=put **host=**<*a.b.c.d*> [**path=**<*flashname*>] [**file=**<*filename*>] [**object=**<*tableobj*>]

cmd=get **host=**<*a.b.c.d*> [**path=**<*flashname*>] [**file=**<*filename*>] [**object=**<*tableobj*>]

cmd=state **host=**<*a.b.c.d*> [**file=**<*filename*>] [**object=**<*tableobj*>]

Note:

Configuration files may contain the current passwords for the Read, Write and Admin Communities. If you use the cmd=put or cmd=state commands to transfer BRICK configuration files to remote hosts, you should also control access to these files for security reasons.

Sending TFTP Files

Once TFTP is setup you can assign "put" to the *biboAdmConfigCmd* object to transmit configuration information stored in flash to a file on a remote host. Only Read-Write information is included in the file.

The TFTP file to be written on the remote host must already exist (and, for UNIX hosts, must be world writable) prior to executing the command.

If problems occur in connection with older BSD based TFTP servers see the Special Note: in Chapter 5.

Usage:

cmd=put [**host=**<*a.b.c.d*>] [**path=**<*flashname*>] [**file=**<*filename*>] [**object=**<*tableobj*>]

Optional arguments:

host	Specifies the IP address of the host to send information to. A hostname can also be used if it can be resolved via DNS. If not specified the address set in ***biboAdmNameServer*** is used by default.
path	Specifies the file in flash RAM to copy data from. If not specified the default flash file is "boot".
file	Specifies the TFTP file to create on the remote host. The file name is relative to the TFTP-boot directory configured on the host. (The default is **C:\BRICK** for PCs running DIME Tools' TFTP Server; or the last field of the tftp entry in **/etc/inetd.conf** on UNIX systems.) The file name defaults to "brick.cf" if <*file*> is not specified.
object	Specifies the table objects(s) to send. Either a specific table, or a complete configuration file can be sent. If no objects are specified, all tables are sent by default.

To retrieve the ***ifTable*** from the flash file "temp" and store the information in file "file.cf" on the host at 192.168.3.4 this command would be used:

```
cmd=put host=192.168.3.4 path=temp file=file.cf ⇨
       object=ifTable
```

Retrieving TFTP Files

You can also retrieve configuration data from remote hosts by assigning "get" to the *biboAdmConfigCmd* object. Once the retrieved file (or table information) is written to flash, the information can then be loaded into memory with cmd=load for it to take effect.

Usage:

cmd=get [**host=**<*a.b.c.d*>] [**path=**<*flashname*>] [**file=**<*filename*> **object=**<*tableobj*>]

Optional arguments:

host Specifies the IP address of the host to retrieve data from. A hostname can also be used if it can be resolved via DNS. If not specified the address set in ***biboAdmNameServer*** is used by default.

path Specifies the file in flash to write data to. If the file already exists in flash its contents are overwritten. The default flash name is "boot".

file Specifies the file on the remote host to retrieve data from. If not specified the TFTP file named "brick.cf" is requested.

object Specifies the object(s) to retrieve. Here, either a specific table or a complete configuration file can be retrieved. If not specified all system tables are retrieved.

For example, using the command

```
cmd=get host=192.168.3.4 path=file.cf file=temp ⇨
        object=ifTable
```

would retrieve the ***ifTable*** from the file *file.cf* on host 192.168.3.4 and save it in a flash file named "temp".

Transmitting State Information

The previously mentioned TFTP commands only send or retrieve variables with Read-Write status. They also send/retrieve information from files stored in flash. Using "cmd=state", you can save all configuration information currently in memory and send the data to a remote TFTP host. This information includes Read-Write AND Read-Only data such as status/accounting information.

The TFTP file to be written on the remote host must already exist (and, for UNIX hosts, must be world writable) prior to executing the command.

If problems occur in connection with older BSD based TFTP servers see the Special Note: in Chapter 5.

Usage:

cmd=state [**host=**<*a.b.c.d*>] [**path=**<*flashname*>] [**file=**<*filename*>] [**object=**<*tableobj*>]

Optional parameters:

host	Specifies the IP address of the host to send data to. If not specified the current value of ***biboAdmNameServer*** is used by default.
file	Specifies the file name on the remote host to write data to (and is relative to the TFTP boot directory on that host). If not specified the default file name is "brick.st".
object	Specifies the table(s) to retrieve data from. If no objects are specified, the contents of all tables are sent.

For example, using

```
cmd=state host=1.2.3.4 file=brick1.st ⇨
    object=system
```

would retrieve all data from the ***system*** table and places it in *"/tftpboot/brick1.st"* on host 1.2.3.4 (if present the file is overwritten).

Tip If you need to contact BinTec support, it is recommended that you have a complete state file available. This would be done with the command shown below.

cmd=state host=<*IP Address*>

where <*IP Address*> identifies a TFTP host you have access to.

Transferring Files with XMODEM via Serial Port

It is possible to load and save configuration files via the serial interface using the protocol XMODEM. Therefore the variable ***file*** is assigned the value **xmodem** or **xmodem-1k**. **xmodem-1k** uses a packet size of 1024 byte (default: 128 byte) and in general reaches a higher throughput. The packet size is defined by the sender so that the value **xmodem-1k** only makes sense on the sending end; on the receiving end it is ignored.

To make use of this new feature you have to access your BRICK from a computer via the serial port and a terminal program.

Getting the Configuration

```
cmd=get file=xmodem path=new_config
```

loads a file received via XMODEM with the name new_config into the flash ROM of the BRICK.

After this command has been started the terminal program must be set to Send (Upload) and the transmission protocol (XMODEM) as well as the source file name and location must be entered. For the time of the file transfer the console cannot be used.

Putting the Configuration

```
cmd=put file=xmodem path=boot
```

sends the BRICK's flash ROM file boot via XMODEM.

After this command has been started the terminal program must be set to Receive (Download) and the transmission protocol (XMODEM) as well as the destination file name and location must be entered. For the time of the file transfer the console cannot be used.

Transmitting State Information

The previously mentioned commands only send or retrieve the configuration files containing variables with Read-Write status. They send/retrieve information from files stored in flash. Using "cmd=state" you can save all configuration information current-

ly in memory. This information includes Read-Write AND Read-Only data such as status/accounting information.

cmd=state file=xmodem

Note:

If you use cmd=put or cmd=state to transfer BBRICK configuration files, you should also control access to these files for security reasons.

When nothing is specified the currently selected baud rate is used for the transfer. The transfer baud rate can be changed by adding @baud to the file variable, e.g.:

```
cmd=put file=xmodem@9600 path=boot
```

Possible baud rates are 1200, 2400, 4800, 9600, 19200, 38400, 57600, 115200. For transmitting data to the BRICK (cmd=get) you should not select a rate higher than 9600. Selecting higher than default baud rates may result in transmission errors. There are no limitations for BIANCA/BRICK-XL/XL2.

In case of transmission errors a syslog is generated.

This feature can only be used via the SNMP shell, not via Setup Tool.

Rebooting the System

The system can also be rebooted via SNMP by assigning "reboot" to the ***biboAdmConfigCmd*** object. This can be used for example, to stop and restart the system remotely from a telnet, isdnlogin, or minipad session.

```
cmd=reboot
```

No additional parameters are required.

4

Chapter Four

ISDN CONNECTIONS ON THE BRICK

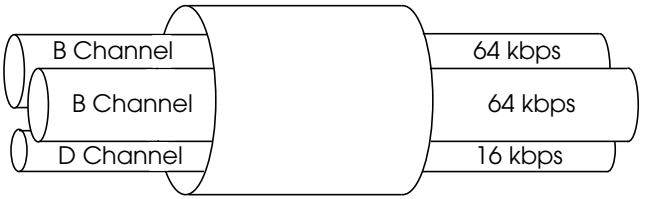
What's Covered?

- ISome background on ISDN
  - B and D Channels
  - ISDN Interfaces
    - Basic Rate Interface
    - Primary Rate Interface
  - Called & Calling Party's Numbers
    - Local Number
  - ISDN Screening Indicator
- Attached ISDN hardware
  - ISDN Auto Configuration
- ISDN Call Dispatching
  - Overview
  - Dispatching Algorithm
    - Routing Service
    - Login Service
    - Pots Service
    - CAPI Service
- ISDN Line Management
  - ShortHold
  - Bandwidth on Demand
  - Multiple Link Support

Some background on ISDN

The term ISDN (Integrated Services Digital Network) was defined by the ITU-T (formerly CCITT) and describes a telecommunications service package supported by telephone companies around the world. As an enhancement to the existing public telephone network, ISDN allows voice, data, video, etc. to be transmitted over existing telephone lines using digital transmission. This allows ISDN users to access multiple services such as telephony, telex, teletex, fax, videotex, and X.25 networking simultaneously from one access point. The ISDN access point, often called subscriber outlet, consists of a standard RJ-45 twisted pair port.

The subscriber outlet can be seen as the user end of a sort of 'digital-pipe' which transfers digital traffic to and from the local telephone company. This digital pipe allows data transfers using a number of channels, commonly known as the B and D channels.



ISDN Basic Rate Interface

B and D Channels

B-channel: The B-channel is used for transferring user data; text, data, voice and still images in full duplex mode. The B-channel can handle data transmission at a rate of 64 kbps.

D-channel: The D-channel's primary function is for signalling between the user equipment (telephone, facsimile, computer, etc.) and the telephone company. The D-channel can handle data transmission at a rate of 16 kbps. In Euro-ISDN the D-channel can also be used for transferring user data.

ISDN Interfaces

The capacity and type of service this digital pipe provides can vary and depends on the type of access you have to the ISDN. The most common types of ISDN interfaces, which are defined by the ITU-T, are the basic rate interface (BRI) and the primary rate interface (PRI). These in turn determine the number of available channels within the pipe and the transfer rates used by each channel.

Basic Rate Interface

An ISDN basic rate interface, or BRI is sometimes called an S_0 interface. It provides two B-channels (64 kbps each) and one D-channel (16 kbps) allowing for a total user data rate of the 144kbps (2 x 64 kbps + 16 kbps). Up to eight end-devices can be connected to an S_0 interface, including telephones, facsimile machines, computers etc.

The network sends control messages over the D-channel to establish connections with the end-devices corresponding to the type of service requested. Two different end-devices can be used simultaneously and independently via a single S_0 interface.

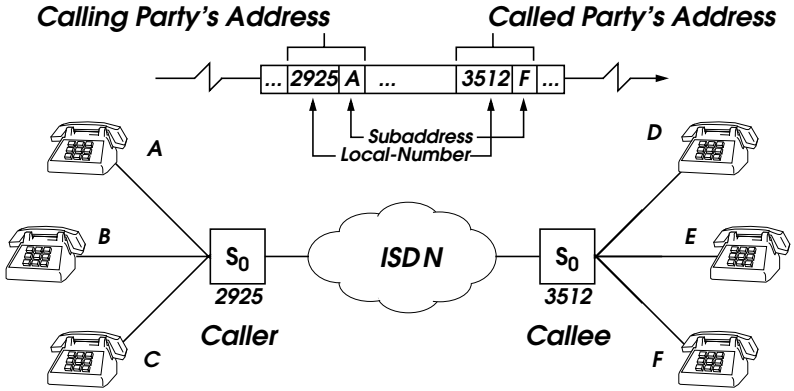
Primary Rate Interface

A primary rate interface, or PRI, is sometimes called an S_{2M} interface. It provides 30 B-channels and one D-channel. As with a BRI the D-channel is used for signalling but since 30 B channels need to be managed in a PRI, the D-channel has a data rate of 64 kbps. This allows for a total user data rate of 1.984 Mbps (31 x 64 kbps).

In North America, a PRI consists of 23 B-channels and one D-channel. The differences between the number of channels are historically based and relates to voice technologies that existed when ISDN was developed.

Called & Calling Party's Numbers

The signalling used in setting up ISDN calls includes information that identifies both the **caller** and **callee** and is referred to as the *Calling Party's Number* and *Called Party's Number* respectively. Both addresses consist of a **Local Number** and an optional **Subaddress.** The calling party and called party numbers are sometimes called *directory numbers* in the ISDN world.



Local Number

Most ISDN basic rate interfaces today come with three separate telephone numbers. These numbers are commonly used to identify specific telecommunications equipment at a subscriber's site. For example, the caller in the diagram above might have the ISDN numbers 2925, 2926, and 2927. Euro-ISDN and National-ISDN in Germany use different names and procedures for identifying separate local numbers.

NOTE: Subaddressing in ISDN should not be confused with the different local numbers available in Euro ISDN and 1TR6 (National ISDN in Germany). Subaddressing is not available in the 1TR6 protocol. In other ISDN protocols subaddresses are optional and usually have to be purchased separately from the ISDN provider.

- **Euro-ISDN**
  In Euro-ISDN, the DSS1 signalling protocol is used. Depending on the type of service arrangement with the local telephone company, the subscriber receives

three or more Local Numbers. These numbers are called MSNs (multiple subscriber number). Additional MSNs can normally be purchased from the ISDN service provider.

- **1TR6** (National ISDN in Germany)
  In Germany, the 1TR6 signalling protocol is used. The ISDN number assigned by the telephone company, e.g. 0911 / 99002, can be extended by appending an additional digit (known as the EAZ, or Endgeräteauswahlziffer) to this number. Up to nine different EAZ numbers (1...9) can be used. EAZ 0 is used as a global, to allow all equipment to receive incoming calls in parallel. This signalling protocol is not supported by PABX-BRICKS and, in any case, will no longer be supported in the new millenium, as it is being replaced by the DSS1.

ISDN Screening Indicator

The ISDN screening indicator is a service provided by ISDN that can be used to test the trustworthiness of the calling party's number. The calling party's number (CPN) reported by an incoming call may have been assigned by the user placing the call or by the telephone switching station.

If the CPN was assigned by the user the switching station may optionally verify this address is correct in order to detect malicious calls. The party (user or network) that assigned the CPN and whether or not the CPN has been verified is reported in ISDN in the Screening Indicator field of the call packet. The values shown below are used and indicate the respective circumstances.

Screening Indicator	CPN assigned by	Status of Calling Party's Number
network	network	The CPN was set by the network. (verification not required)
user-verified	user	The CPN was set by the user and was verified by the network.
user	user	The CPN was set by the user but no verification was attempted.
user-failed	user	The CPN was set by the user and verification of the number failed.

Attached ISDN hardware

ISDN Auto Configuration

The ISDN auto configure procedure attempts to verify:

1. The presence of each ISDN interface.
2. Which type of D channel protocol is used.
3. Which TEI procedure is used.

Normally, the BRICK attempts to configure its ISDN interface(s) automatically at boot time (see Turning Off Auto Configuration). If your ISDN module is installed, the auto configuration process is started once the module is connected to your subscriber outlet. The configuration process can also be started manually while the system is running (see Restarting Auto Configuration).

Once the auto configure process is complete (see Verifying Auto Configuration) the BRICK initializes a protocol stack for each D channel. The results of the auto configure process are then written to the ***isdnStkTable*** which lists the attributes of each ISDN stack. Access to the ISDN is possible once the following objects are defined:

isdnStkTable **Field**	**Must be:**
ProtocolProfile	dss1 **or** dtr6
Configuration	point_to_point **or** point_to_multipoint
Status	loaded

Verifying Auto Configuration

You can verify whether the auto configuration procedure was successful. First, display the status of the auto configure process by displaying the ***isdnIfTable***. If the ***AutoconfigState*** field is set to **done**, then the auto configure procedure is complete.

Next, verify the operational status of the interface by viewing the ***isdnStkTable***. If the ***Status*** field is set to **loaded** then auto configuration was also successful. The interface is ready to accept connections.

Turning Off Auto Configuration

As long as *isdnIfAutoconfig* is set to **on** (the default), auto configuration will be performed. If set to **off** then information for the respective interface will not be configured, but will be loaded from the *isdnStkTable* instead.

Restarting Auto Configuration

If auto configuration was not successful you can restart the procedure by assigning **start** to the *isdnIfAutoconfigState* field of the isdnIfTable .

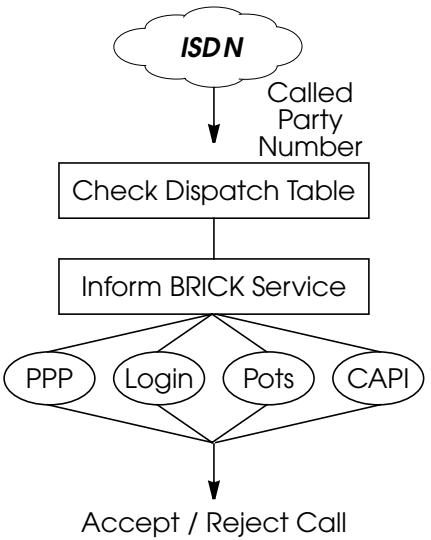
```
isdnIfAutoconfigState:0=start
```

ISDN Call Dispatching

Overview

The BRICK uses an internal Dispatching Algorithm to dispatch incoming ISDN calls to various services based on the Called Party Number, or CPN, signalled by the ISDN. Currently, BRICK services include:

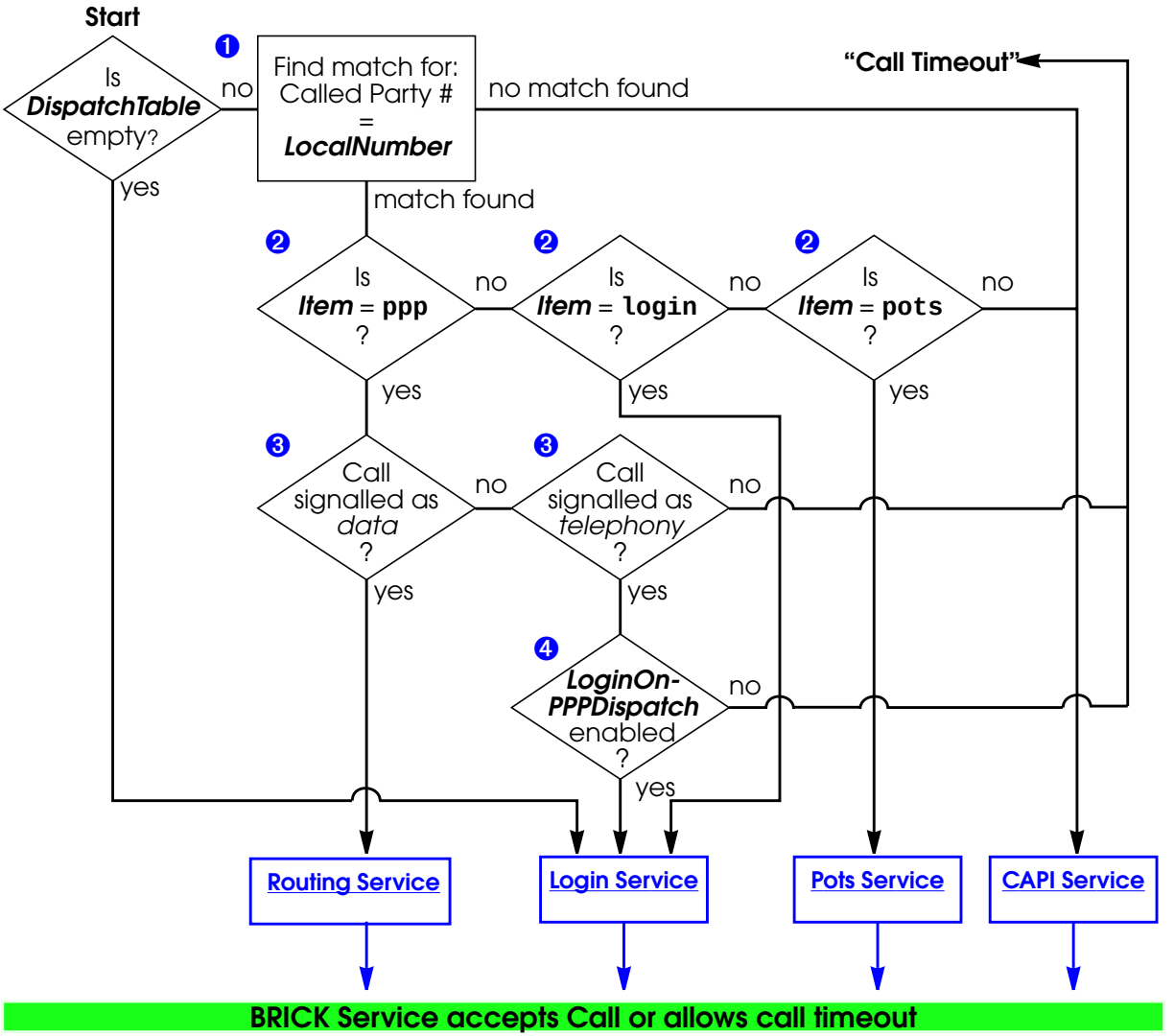
- Routing Service — The PPP service is the BRICK's main routing service. This service is used for incoming data calls for dialup network connections from ISDN WAN partners.
- Login Service — The login service provides access to the SNMP shell.
- Pots Service — The pots service is only available on the V!CAS and is for calls that need to be routed to attached analog devices (V!CAS POTS ports A and B).
- CAPI Service — The CAPI service is used for incoming calls from remote CAPI applications (version 1.1 or 2.0) that need to connect to CAPI application running on a workstation on the BRICK's LAN.



The basic procedure for dispatching incoming calls shown here simply means that when an incoming call is received, the BRICK then searches the ***isdnDispatchTable*** for an entry that matches the CPN. If a match is found the call is given to the appropriate service which may decide to accept or reject the call based on other information relating to the call. If no match is found the call is given to the CAPI service which informs CAPI applications on the LAN of the call.
These decisions are described in further detail in the following sections. Note that the dispatching table is also used for outgoing calls too. This is covered in the section Outgoing Calls.

Dispatching Algorithm

This diagram shows the initial steps used to dispatch incoming calls on the BRICK. Information relating to each step is listed on the following page. Additional steps taken by the respective services are described separately.



❶

Here, the ***isdnDispatchTable*** is searched for a matching entry. Note that since this is an incoming call, only entries with ***Direction*** = both or incoming are valid in this context. A match is found if the Called Party's Number matches the ***LocalNumber*** field. It is important that each MSN is mapped to no more than one service.

❷

The ***Item*** field of the matched entry determines which BRICK service is informed of the call.

❸

Note that the ISDN may signal a call (just another term for identifying the calls type) as being a data or a telephony call. This step has been implemented for sites that only have one MSN. See step 4.

❹

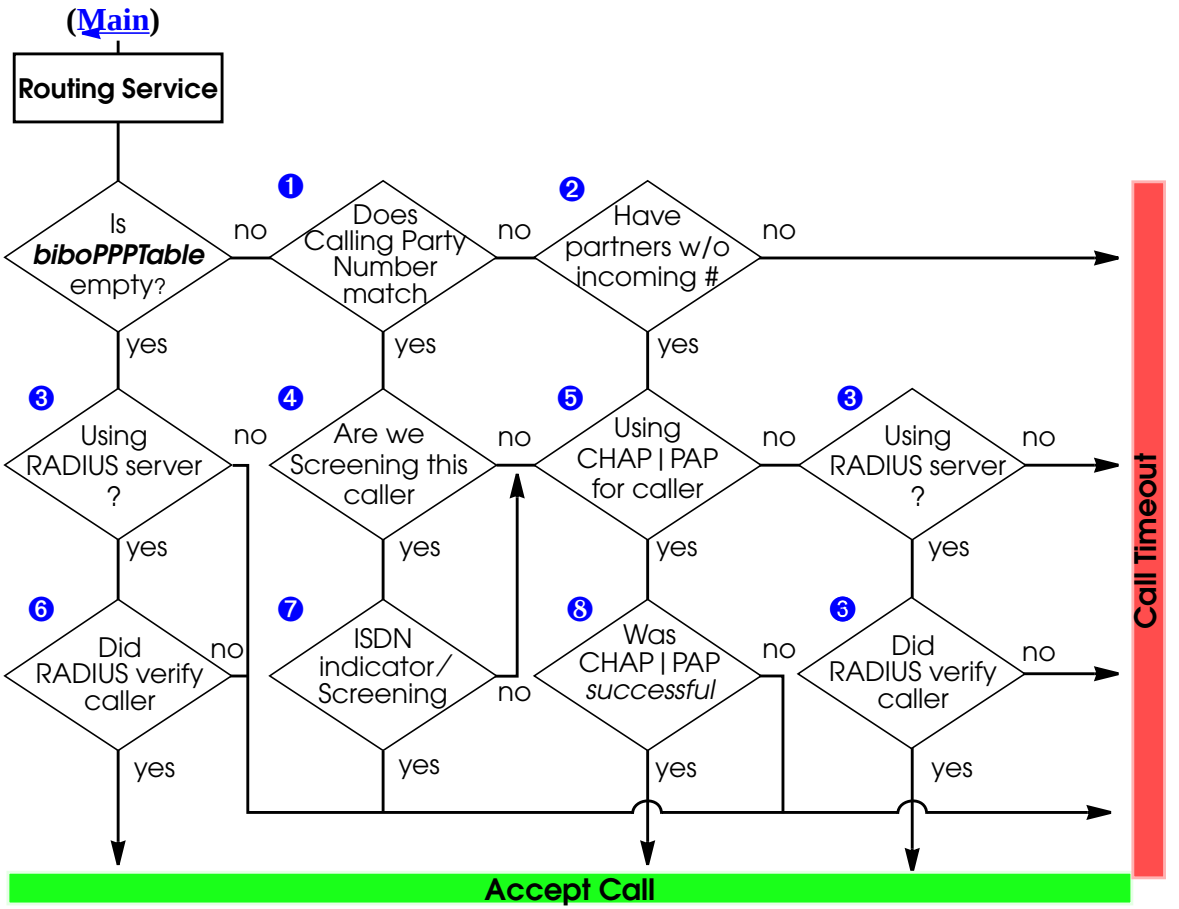
This additional step has been implemented for sites that only have one MSN. Since these sites will have to use their sole MSN for the routing service this step allows them to dispatch calls to the login service using the ***isdnLoginOnPPPDispatch*** variable.

The isdnlogin command (from the SNMP shell) can be used from a remote BRICK to establish an ISDN call to a BRICK with one MSN (and appropriately configured) using the servicename "telephony".

`isdnlogin` <*telephone number*> **`telephony`**

Routing Service

The Routing service decides whether to accept or reject the call based on the diagram shown below.



❶

Here, the Calling Party's Number transmitted via the ISDN is compared to each entry's ***Number*** field in the ***biboDialTable***. This is the first step of "OUTBAND" authentica-

tion, also known as Calling Line ID. Note that even if a match is found the caller still has not incurred any charges at this point.

❷

If the Calling Party's Number couldn't be matched in the ***DialTable*** the BRICK checks to see if any WAN partner's exist (***biboPPPTable***) that do not have an incoming number (***biboDialDirection*** = `both` or `incoming`).

❸

If a RADIUS server is configured in ***biboAdmRadiusServer*** this step resolves as yes. The call is then initially accepted (charges are incurred by the caller) if it hasn't already been, and the RADIUS server is consulted.

❹

The Screening field from the matched ***DialTable*** entry from step 1 determines whether screening should be performed for calls from this number. If ***Screening*** for this entry is set to `dont_care` the screening feature is not being used. Screening is the second step of OUTBAND authentication; meaning that ISDN charges for the caller still have not been incurred.

Background information on ISDN Screening is covered the section ISDN Screening Indicator.

❺

WAN partners configured to use CHAP and/or PAP authentication are identified in the ***biboPPPTable*** by the ***Authentication*** field which will be set to either; `pap`, `chap`, or `both`.

Once the dispatching algorithm reaches this step, the ISDN call is initially accepted to perform INBAND authentication.

❻

If the RADIUS server was able to verify the caller, the BRICK accepts the call and establishes the network connection according to the parameters provided by the RADIUS server. Otherwise, the call is disconnected.

❼

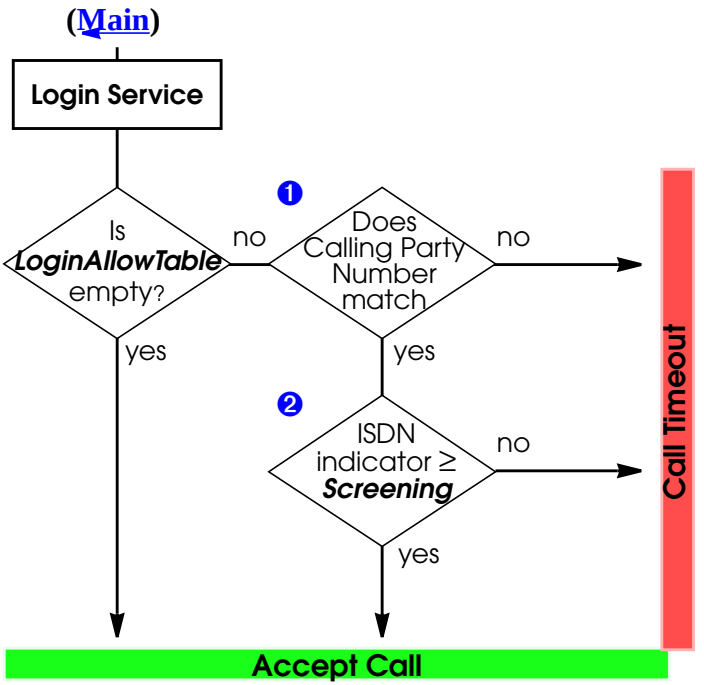
When screening incoming calls, the ***biboDialScreening*** variable is compared to the screening indicator provided by the ISDN. The value provided by the ISDN must be greater than or equal to ***biboDialScreening***.

❽

The last step for WAN partners that must authenticate via CHAP or PAP.

Login Service

The Login Service may accept or reject an incoming call based on the diagram shown below.



❶

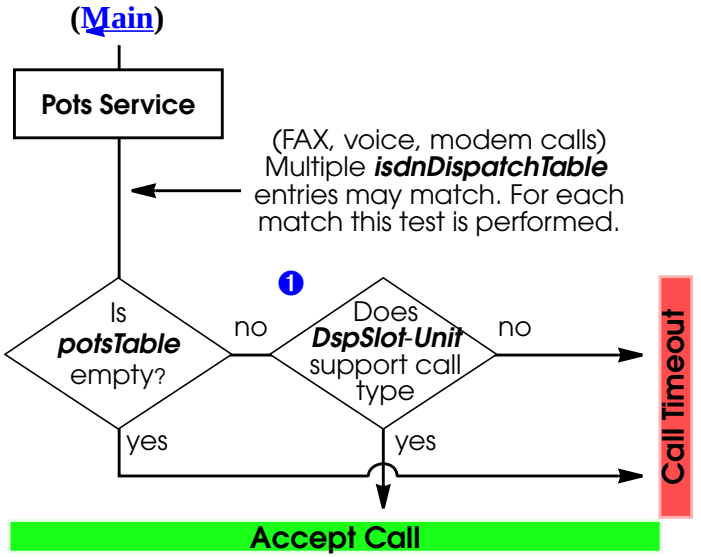
Here, the ***isdnLoginAllowTable*** is searched for a matching entry. A match is found by comparing the Calling Party's Number with the ***Number*** fields of each entry. Note that the number field supports wildcard characters and multiple entries may match an incoming call. A match without wildcards is always used before a match with wildcards.

❷

Once a match is found the value of the ***isdnLoginAllowScreening*** field is compared with the screening indicator provided by the call setup packet. The call is accepted if the indicator (from the setup packet) is greater than or equal to the ***Screening*** field.

Pots Service

The Pots Service on the V!CAS may accept or reject an incoming call based on the diagram shown below. The Pots service is currently only available on the V!CAS.

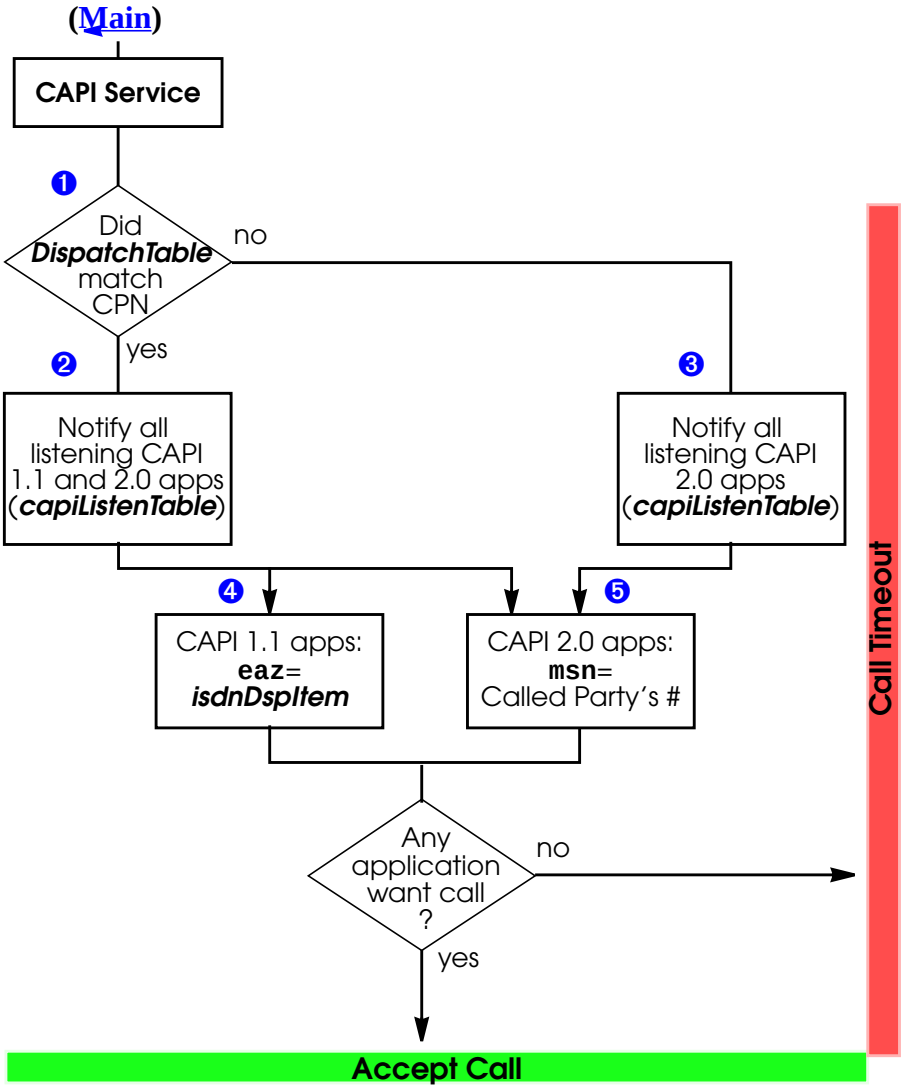


1

The Slot and Unit fields of the matched isdnDispatchTable entry determine the destination device for the call. The corresponding device entry is located in the *potsIfTable*. The Type field there, determines which types of calls the device supports. See the section on POTS Interfaces in Chapter 12 for more information on the pots devices.

CAPI Service

The CAPI service on the BRICK must inform registered CAPI clients of incoming calls. Depending on the Dispatch Table, the type of call, and the registered applications, the CAPI service accepts or rejects calls as follows.



❶

The ***isdnDispatchTable*** (***LocalNumber*** field) is searched for an entry that matches the called party's number contained in the ISDN call setup packet.

❷

If a match was found from the previous step, the ***Item*** field must be set to one of the values `eaz0` through `eaz9`. (see the initial dispatching diagram). In this step the BRICK checks the contents of the ***capiListenTable*** to see which CAPI applications are listening for incoming calls. Listening version 1.1 and version 2.0 applications will be notified of the incoming call.

A brief overview of how the listening process is covered in the section The Remote CAPI in Chapter 11.

❸

If the ***isdnDispatchTable*** contains entries but no matches were found (see the initial dispatching diagram) the call defaults to the CAPI service. In this step the BRICK checks the contents of the ***capiListenTable*** for listening applications. Since no EAZ→MSN mapping is involved here, only CAPI 2.0 are notified of the incoming call. CAPI 2.0 applications use MSNs.

A brief overview of how the listening process is covered in the section The Remote CAPI in Chapter 11.

❹

CAPI 1.1 applications use EAZs. When notifying CAPI 1.1 applications of a call, the EAZ value is taken from the ***isdnDspItem*** field.

❺

CAPI 2.0 applications use MSNs. When notifying CAPI 2.0 applications, the Called Party's Number from the call setup packet (which will be the same as the ***LocalNumber*** field if an ***isdnDispatchTable*** entry was matched) is sent as the MSN.

Outgoing Calls

For outgoing calls from CAPI 1.1 applications, the BRICK compares the EAZ transmitted by the CAPI 1.1 application with the contents of the ***isdnDspItem*** object. Once a match is found, the BRICK uses the respective ***isdnDspLocalNumber*** (and ***isdnLocalSubaddress*** if set) object as the "Calling Party's Address".

ISDN Line Management

ShortHold

To help minimize ISDN charges the ShortHold mechanism is available. ShortHold closes down unneeded dialup connections when there is no traffic to be transmitted for a specific time period. Short hold is enabled by default for all dialup partner interfaces. Two types of Short Hold mechanisms are available on the BIRCK; Static Short Hold and Dynamic Short Hold.

With ShortHold you can control the amount of time to wait before closing all remaining B-channel(s). This means that when no packets are being sent or received, the system will keep a minimum number of channels open until the ShortHold timer expires.

Bandwidth on Demand

By measuring current line usage at regular intervals, additional ISDN channels for a particular connection can be opened or closed when needed.

The initiating ISDN caller assumes control for line monitoring. Every five seconds the current throughput rate is measured. When throughput rises above an upper bound an additional channel is opened. If the throughput rate drops below a lower bound unneeded channels are closed automatically. The upper and lower bounds are defined as follows:

- **Upper-bound**: The most recently opened channel is at 90% usage.
- **Lower-bound**: The most recently opened channel is at 0% usage and the channel before that is at 80% usage.

Multiple Link Support

... ISDN partners to be run over multiple channels. By dynamically allocating bandwidth (opening or closing of additional channels) higher throughput rates can be

5

Chapter Five

SYSTEM ADMINISTRATION ON THE BRICK

What's Covered?

- System Logging on the BRICK
- Gathering Accounting Information
  - ISDN Accounting Information
  - IP Accounting Information
- Logging with Remote LogHosts
- Remote SNMP Administration
  - Traps
- Web Based Monitoring
- User Accounts
- Other Passwords
- System Software Updates
- BOOT Options on the BRICK
  - The BOOTmonitor
  - Booting via BootP
  - BootP Relay Agent
- Other System Administration Tasks
  - Setting Up a BootP Server
  - Setting up a TFTP Server
  - Setting Up a syslog Daemon

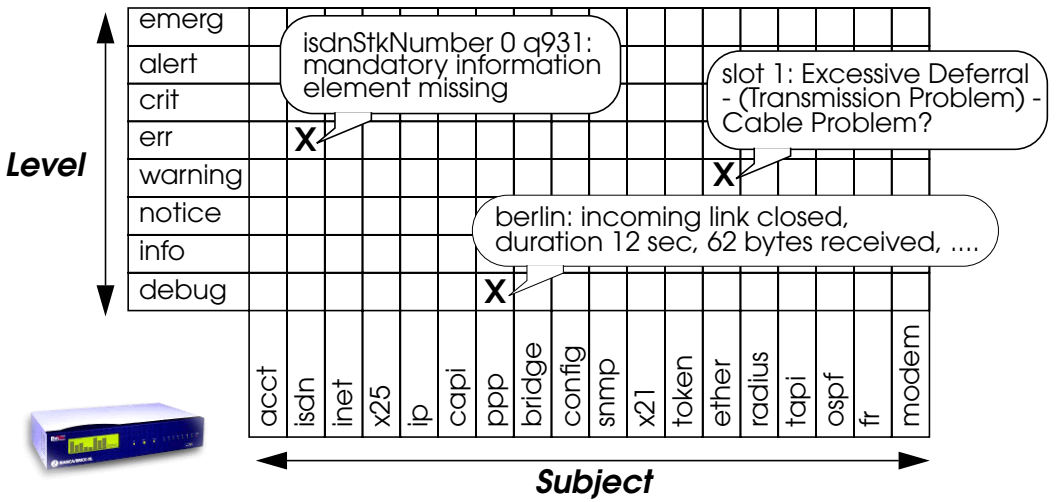
System Logging on the BRICK

During normal operation various messages may occasionally be generated on the BRICK by its various subsystems (ISDN, PPP, X.25, MODEM, etc.). These messages, called syslog messages, are generated in response to error conditions or other events that may occur while the system is running.

A syslog message is a text string consisting of four pieces of information relating to the event that occured. Syslog messages are stored locally in the BRICK's ***biboAdmSyslogTable***. A limited number of messages are saved here (defined by the value of the ***biboAdmSyslogMaxEntries*** object, default is 20); each time the system reboots existing messages are lost.

The ***biboAdmSyslogTable*** consists of the following fields.

TimeStamp	A date string of the format: **`MM/DD/YY HH:MM:SS`** that identifies the date and time the message was generated.
Level	The severity of the event; i.e., the higher the level the more important the message is considered (see below).
Messsage	The actual text of the message. The text attempts to describe the circumstances relating to the event.
Subject	The internal software subsystem that generated the message.



Recent system messages can be displayed from the SNMP shell at any time by entering **`message`** at the shell prompt.

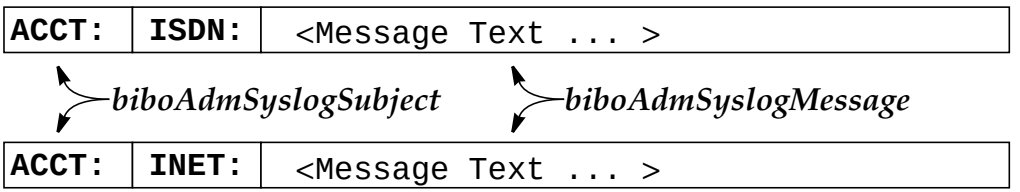
Accounting Messages and System Messages

Syslog messages fall into two categories; Accounting messages and System messages. Accounting messages are generated by the **acct** subsystem. (See the ***Subject*** field of the ***biboAdmSyslogTable*** above.) System messages are generated by any of the other BRICK subsystems which, depending on the current license(s) installed on the system, may include:

```
isdn    inet    x25     ipx     capi    ppp
bridge  config  snmp    x21     token   ether
radius  tapi    ospf    fr      modem
```

Accounting Messages

Accounting messages are used to report accounting information relating to either an ISDN connection or an IP session that was closed/routed over the BRICK. Accounting messages are identified (in the ***biboAdmSyslogTable*** or in a remote file on a LogHost where syslog messages are being sent) by an initial **ACCT:** tag in the text of the message. For ISDN messages, the **ISDN:** tag immediately follows; for IP accounting messages **INET:** follows.



ISDN Accounting Messages

An ISDN accounting message contains information regarding an ISDN call that was either placed or received by the BRICK. Details for both successful and unsuccessful ISDN outgoing calls are reported here.

Note

If a B-channel is used to its full capacity for at least three days, ISDN Accounting information can overload resulting in the subsequent sending of erroneous accounting messages.

The content and format of ISDN accounting messages vary according to the special formatting tags contained in the ***isdnAccountingTemplate***. A list of possible format tags that can be used in the accounting template and their meanings are shown below.

Format Tag	Meaning
%S	Date the connection opened; in DD.MM.YY format.
%s	Time the connection was established: in HH:MM:SS format
%R	Date the connection closed; in DD.MM.YY format.
%r	Time the connection was closed: in HH:MM:SS format.
%d	The duration of the connection in seconds.
%y	Total number of bytes received over the connection.
%Y	Total number of bytes sent over the connection.
%g	Total packets received over the connection
%G	Total packets sent over the connection.
%c	Total number of charging units (value) incurred for the connection.
%C	Total number of charging units (string) incurred for the connection.
%n	The call's direction; either incoming or outgoing.
%Z	The local address (Calling or Called party's number, see **%n**).
%z	The local subaddress (Calling or Called party's number, see **%n**).
%T	The remote address (Calling or Called party's number **%n**).
%t	The remote subaddress (Calling or Called party's number **%n**).
%i	Service indicator and additional information for the call.

Format Tag	Meaning
%b	Bearer capability for the call.
%l	Low layer capability for the call.
%h	High layer capability for the call.
%u	DSS1 error cause, if applicable.
%U	1TR6 error cause, if applicable.
%L	Local (BRICK internal) error cause.
%F	Call reference (BRICK internal).
%I	Information about the BRICK subsystem the call was given to.

The default accounting template setting contains the following tags:

%S,%s,%r,%d,%y,%Y,%g,%G,%C,%n,%Z,%T,%i,%u,%L

This template produces accounting messages similar to the following.

ISDN:18.08.1997,13:53:19,13:53:34,12,1096,1875,33,33,1Units,O,2,003039988452,7/0,9F,0

Changing the ISDN Accounting Template

The accounting template can be changed to meet your particular needs. As shown above the comma character is used as the default delimiter, separating each data field. However, since the ***isdnAccountingTemplate*** is a quoted string arbitrary words and characters may be added as needed.

This may be useful for sites forwarding accounting messages to remote UNIX loghosts and performing post-processing (via grep or other shell scripts). Setting the accounting template to the value:

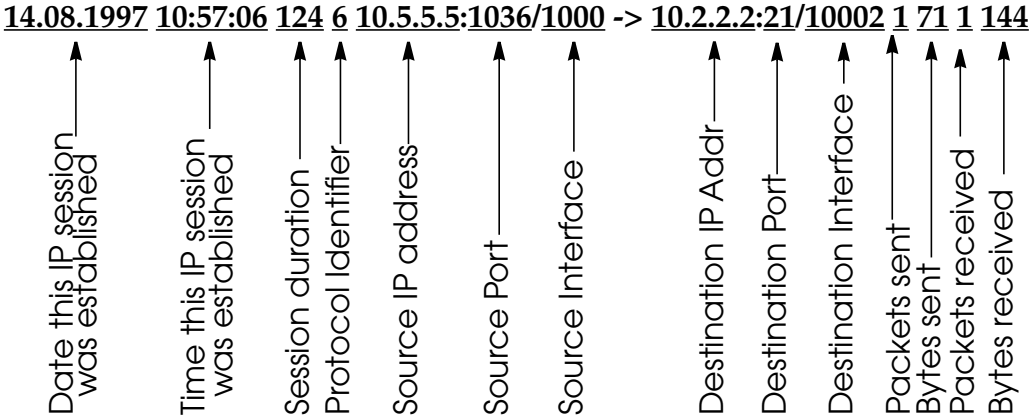
"%S## LinkUp@%s-Down@%r (Called %n->to %T) %c charging units"

would result in less informative, more readable messages similar to:

```
18.08.1997## LinkUp@17:36:08-Down@17:36:10(Called Out->to 254)      0 charging units
18.08.1997## LinkUp@17:36:08-Down@17:36:10(Called in->to 187)       7 charging units
18.08.1997## LinkUp@17:36:08-Down@17:36:10(Called Out->to 794)      5 charging units
18.08.1997## LinkUp@17:36:08-Down@17:36:10(Called Out->to 234)      4 charging units
```

IP Accounting Messages

IP accounting messages contain information for a specific IP session that was routed over the BRICK. In contrast to ISDN accounting messages, IP accounting messages have a fixed format and can't be changed. A sample IP accounting message showing the respective fields is shown below.



IP accounting messages are only generated for IP sessions routed over IP interfaces for which accounting has been enabled. This is done by settig the respective ***ipExtIfAccounting*** variable in the ***ipExtIfTable*** is set to **on**.

Once accounting for an interface is turned on, active IP sessions routed over the interface appear in the ***ipSessionTable***. Once a session closes, either by disconnection or timeout, an accounting message is generated and is written to the ***biboAdmSyslogTable***.

System Messages

System messages are generated by BRICK system software subsystems in response to certain errors or events. Recall that all syslog messages include a BRICK subsystem tag

at the beginning of the message text. System messages are identified by any subsystem tag other than the **ACCT**: tag .

The most common system messages are shown in Appendix E.

Gathering Accounting Information

Accounting information relating to active or closed ISDN connections or IP sessions on the BRICK can be queried locally on the BRICK via various system tables or logged to remote hosts using the syslog protocol.

ISDN Accounting Information

ISDN accounting messages contain information about ISDN calls that was either placed or received by the BRICK.

Credits Based Accounting System

With dial-up WAN connections it may occur that charges rise because of configuration errors. The Credits Based Accounting System gives BRICK administrators the ability to control charges. It allows the BRICK administrator to watch and limit the number of connections, the connection time and the accounted charges of every subsystem during a specified period of time. If the limit is exceeded the BRICK can't make further connections in that period of time. Syslog messages give you information about credits, when the 90% or 100% mark for each limit and each subsystem is reached. Also, each time a call is rejected a syslog message is generated.

The new ***isdnCreditsTable*** controls this feature, it is described in the current MIB Reference http://www.bintec.de/download/brick/doku/mibref/index.html.

The Credits Based Accounting System can also be configured via Setup Tool: in the main menu over **ISDN** to manage and activate the system; and over **Monitoring and Debugging** to monitor the incoming and outgoing connections and accounted charges.

Tracking Current ISDN Connections

Statistics for current ISDN calls are stored in the ***isdnCallTable***. As long as the call is active, the corresponding fields in this table are updated. Once an ISDN call is closed, or disconnected, the ***isdnCallTable*** entry is removed and a new entry is created (using the data from the ***isdnCallTable*** entry) in the ***isdnCallHistoryTable***.

To show how these table entries are created/removed, we'll establish a loopbacked ISDN connection to our BRICK using our own ISDN telephone number (143) in the example below. This assumes that incoming call dispatching has been configured allowing calls to 143 to be given to the login service.

Using the **isdnlogin** program we place the call and login as **admin**.

```
mybrick: system> isdnlogin 143
Trying...
Establishing B-channel...
Connected to 143

Connected to BIANCA/BRICK-XS, mybrick, Germany

Welcome to BIANCA/BRICK-XS version V.4.5 Rev.3 from 97/08/01 00:00:00
systemname is mybrick, location Germany

Login: admin
Password:

mybrick: >
```

Then we simply display the ***isdnCallTable*** to see the details of active ISDN connections.

```
mybrick: > isdnCallTable
inx StkNumber(*ro)          Type(*ro)           Reference(*ro)
    Age(ro)                 State(rw)           IsdnIfIndex(ro)
    Channel(ro)             DspItem(ro)         RemoteNumber(ro)
    RemoteSubaddress(ro)    LocalNumber(ro)     LocalSubaddress(ro)
    ServiceIndicator(ro)    AddInfo(ro)         BC(ro)
    LLC(ro)                 HLC(ro)             Charge(ro)
    ReceivedPackets(ro)     ReceivedOctets(ro)  ReceivedErrors(ro)
    TransmitPackets(ro)     TransmitOctets(ro)  TransmitErrors(ro)
    ChargeInfo(ro)          Screening(ro)       Info(ro)

 00 0                       outgoing            4
    0 00:26:30.00           active              2000
    1                       login               "143"

     data_transfer          0                   88:90
                                                0
    553                     2754                0
    542                     8357                0
                            undefined           "isdnlogin"

 01 0                        incoming           2
    0 00:26:30.00           active              2000
    2                       eaz3
                            "3"
    data_transfer           0                   88:90
                                                0
    558                     2834                0
    572                     9183                0
                            undefined           "isdnlogind"
mybrick:isdnCallTable> exit
```

Since we placed a loopbacked call by calling our own ISDN number a separate entry is present for both the incoming and the outgoing call.

The ***Type*** field (shown above) identifies the direction of the call. Details of the ISDN call are contained in the respective fields most of which are self explanatory. For information regarding the meanings of specific fields refer to the MIB reference contained on the Companion CD.

We can terminate the ISDN connection by ending the isdnlogin session started previously. The ***isdnCallTable*** entry is dismissed and a new ***isdnCallHistoryTable*** entry is

created as shown below. Again, since an incoming and an outgoing call was registered, two entries are added to the ***isdnCallHistoryTable***.

```
mybrick: > isdnCallHistoryTable
inx StkNumber(*ro)          Type(*ro)           Time(ro)
    Duration(ro)            IsdnIfIndex(ro)     Channel(ro)
    DspItem(ro)             RemoteNumber(ro)    RemoteSubaddress(ro)
    LocalNumber(ro)         LocalSubaddress(ro) ServiceIndicator(ro)
    AddInfo(ro)             BC(ro)              LLC(ro)
    HLC(ro)                 Charge(ro)          DSS1Cause(ro)
    1TR6Cause(ro)           LocalCause(ro)      ChargeInfo(ro)
    Screening(ro)           Info(ro)

 00 0                       incoming            08/19/97 13:28:25
    39                      2000                2
    eaz3
    "3"                                         data_transfer
    0                       88:90
                            0                   0x9f
    0x80                    0
    undefined               "isdnlogind"
 01 0                       outgoing            i08/19/97 13:28:25
    39                      2000                1
    login                   "143"
                                                data_transfer
    0                       88:90
                            0                   0x9f
    0x80                    0
    undefined               "isdnlogin"

mybrick: isdnCallHistoryTable>
```

Note:

The number of entries in the ***isdnCallHistoryTable*** is limited to the value set in the ***isdnHistoryMaxEntries*** object. By default information regarding the last 20 ISDN calls are saved with older entries being dismissed as newer entries are added.

Most fields shown above are self explanatory. For meanings of the ***DSS1Cause***, ***1TR6Cause***, and ***LocalCause*** fields, refer to Appendix C.

For descriptions regarding the meanings of individual fields in the isdnCallHistoryTable see the BRICK MIB Reference contained on the Companion CD.

Logging ISDN Accounting Information to LogHosts

ISDN accounting messages can be forwarded to remote hosts for storage or post processing. This is done by configuring the remote host as a LogHost on the BRICK in the ***biboAdmLogHostTable.*** LogHosts may include PCs running *DIME Tools Syslog Daemon* program (see: BRICKware for Windows) or a UNIX workstation where the syslog daemon is appropriately configured (see: Setting Up a syslog Daemon).

To configure the LogHost on the BRICK refer to the section on: Logging with Remote LogHosts.

Note:

When configuring LogHosts for accounting information ALL accounting information (both ISDN and IP accounting messages) will be sent to this host.

IP Accounting Information

IP accounting messages contain information about a specific IP session routed over the BRICK. Recall that IP accounting messages are only generated for IP sessions that are routed over interfaces for which IP accounting has been enabled in the ***ipExtIfTable***.

Tracking Active IP Sessions

Statistics for active IP sessions routed over BRICK interfaces (again, interfaces for which IP accounting is enabled) can be seen in the ***ipSessionTable***. Once an IP session closes this entry is removed and a IP accounting message is generated and saved to the ***biboAdmSyslogTable***.

The SNMP session shown below displays the respective table entries that might be created for an FTP session between a host on the BRICK's LAN (***ifIndex*** = 1000 IP Address = 192.168.2.2) and a remote host via a dial-up link (***ifIndex*** = 10002 IP Address = 10.5.5.5).

```
mybrick: > ipSessionTable

inx SrcAddr(*ro)        SrcPort(*ro)        DstAddr(*ro)        DstPort(*ro)
    OutPkts(ro)         OutOctets(ro)       InPkts(ro)          InOctets(ro)
    Protocol(*ro)       Age(ro)             Idle(ro)            SrcIfIndex(ro)
    DstIfIndex(ro)

 00 192.168.2.2         1224                10.5.5.5            21
    45                  1860                28                  1570
    tcp                 0 00:00:10.00       0 00:00:00.00       1000
    10002

mybrick: ipSessionTable>
```

Once the session closes an entry is made to the ***biboAdmSyslogTable*** and if applicable, a message is sent to the configured LogHost(s). When displaying the ***biboAdmSyslogTable*** only the first few characters of the message text is diaplayed. To see the full text enter the **`message`** command.

```
mybrick: > biboAdmSyslogTable

inx TimeStamp(*ro)      Level(*ro)      Message(ro)         Subject(ro)

 00 01/01/70  0:00:09   err             "TIMED: no respon   inet
 01 08/19/97 19:07:35   info            "INET: 19.08.1997   acct

mybrick: ipSessionTable>message

00  "TIMED: no response"
01  "INET: 19.08.1997 18:55:25 709 6 192.168.2.2:1224/1000 -> 10.5.5.5:21/10002 61 2506 41
2380"

mybrick: ipSessionTable>
```

Logging IP Session Information to LogHosts

IP accounting messages can be forwarded to remote hosts configured to accept syslog messages. Such hosts may include PCs running the included *DIME Tools Syslog Daemon* program (see: BRICKware for Windows) or a UNIX workstation where the syslog daemon is appropriately configured (see: Setting Up a syslog Daemon).

To configure the LogHost on the BRICK refer to the section on: Logging with Remote LogHosts.

Note:

When configuring LogHosts on the BRICK for accounting information ALL accounting information (both ISDN and IP accounting messages) will be sent to this host.

Logging with Remote LogHosts

LogHosts are configured on the BRICK in the ***biboAdmLogHostTable***. This table consists of four fields that define the following attributes for the LogHost.

Addr	The IP address of the host to send the syslog message to.
Level	The level of syslog messages to send to this host. This is a minimum level; setting this object to level X sends all messages with levels ≥ X (See: System Logging on the BRICK).
Facility	This is the syslog facility on the LogHost the BRICK sends the message to. This is only required for UNIX LogHosts.
Type	The type (either **`system`**, **`accounting`**, or **`all`**) of syslog messages to send to this host. System and accounting messages are described here, all include both types.

LogHosts configured on the BRICK must be configured to accept messages via the syslog protocol. For PCs the *DIME Tools Syslog Daemon* can be used. For UNIX workstations, the syslogd must be be properly configured and running (see: Setting Up a syslog Daemon).

The BRICK always uses the UDP port 514 for sending syslog messages.

A simple LogHost setup involving one one remote host is shown below. In this example accounting messages, both ISDN and IP, and system messages with levels ≥ **`err`** are sent to this host.

Note:

Because of cost considerations it is generally not a good idea to configure LogHosts that are only accessible via ISDN DialUp links.

Since we want to keep our accounting and system warning messages in separate files on the remote LogHost we need to make two entries in the ***biboAdmLogHostTable***.

```
mybrick: > biboAdmLogHostTable

inx Addr(*rw)          Level(-rw)      Facility(rw)        Type(rw)

mybrick: biboAdmLogHostTable> Addr=192.168.5.99 Level=info Facility=local0 Type=acct
mybrick: biboAdmLogHostTable> Addr=192.168.5.99 Level=err Facility=local1 Type=system

mybrick: biboAdmLogHostTable> biboAdmLogHostTable

inx Addr(*rw)          Level(-rw)      Facility(rw)        Type(rw)

00  192.168.5.99       info            local0              acct
01  192.168.5.99       err             local1              system
mybrick: biboAdmLogHostTable>
```

Note:

Accounting messages are generated at the Level=info. If you configure a log host for accounting me ssages (Type=acct) and specify a level higher than info no messages will be sent to the LogHost.

Assuming our UNIX LogHost was configured to accept these syslog messages via the **`local0`** and **`local1`** facilities and save the information to the **`/var/adm/mybrick.acct`** and **`/var/adm/mybrick.system`** files respectively, we might see the following information accumultate there.

`/var/adm/mybrick.acct`

```
Aug 14 11:19:26 mybrick ACCT: INET: 14.08.1997 11:18:46 1 6
                10.2.2.6:2855/4000->10.4.5.8:25/10002 30 16 1000
Aug 14 11:24:48 mybrick ACCT: ISDN: 14.08.1997,11:24:08,11:24:22,
                12, 1185, 2715,37,37,1 Units,O,2,7834,7/0,9F,0
```

`/var/adm/mybrick.system`

```
Aug 14 11:23:54 mybrick ISDN: isdnStkNumber 0 q931:
                information element missing
```

The initial date and time strings at the beginning of the message are set by the local host (or PC). They reflect the date and time the message was received and may not correspond to the actual time of the system event.

Remote SNMP Administration

Object Identifiers (OIDs)

All OIDs of all MIB variables have the same structural form.

.1.3.6.x.x.x.x .y.y.y.y .i

.1.3.6.x.x.x.x : is the OID of the variable according to the MIB description file

.y.y.y.y : is the specific OID part for the unambiguous identification of a variable in several rows of a dynamic table (non-existent in static tables). It consists of the contents of all index variables (*variables), which are mostly unambiguous by row.

For tables where this is not the case (e.g. ipRouteTable), the following index is required for purposes of clarity.

.i : is a continuing index (always 0 for static tables) not the same as the ´inx´on the Command Line.

The Raw-Mode (numerical form) command x toggles Raw-Mode on and off. After entering the command, the shell reports which mode it is entering. By default Raw-Mode is off from the SNMP shell.

Traps

Standard and Enterprise-Specific Traps

To report asynchronous events to a management station (trap host) the BRICK can send traps. Asynchronous events means e.g. the change of a MIB variable, which may require attention. Traps are differentiated into Standard Traps and Enterprise-Specific Traps.

Standard Traps report the events “coldStart, warmStart, linkDown, linkUp and authentificationFailure” and are sent by default when a trap host is defined or trap broadcasting is turned on.

coldStart	Reboot of the BRICK

warmStart	Reboot of the BRICK
linkDown	Disconnection of a link. Change of the variable ***ifOperStatus*** to the value ***down*** or ***dormant*** (hardware and software interfaces).
linkUp	Establishment of a connection. Change of the variable ***ifOperStatus*** to the value ***up*** (hardware and software interfaces).
authenticationFailure	An SNMP authentication failure, i.e. SNMP request with wrong password.

Enterprise-Specific Traps can be defined by the user. To define a trap object the user must assign a MIB variable (object identifier in dot format or string) to the variable ***biboATrpObj*** in the ***biboAdmUserTrapTable***.

Only certain variables, which could contain important changes, can be trapped. Counters can not be trapped.

Traps can either be broadcasted to the local LAN or be sent to a defined trap host. Trap hosts can be configured in the ***biboAdmTrapHostTable***.

Broadcasting traps into the LAN can be configured with the variable ***biboAdmTrapBrdCast*** in the ***adminTable***, where also the ***TrapPort*** (default: 162) and the ***TrapCommunity*** (default: "snmp-Trap") can be adjusted.

The following two examples explain the structure of trap packets, which are ASN1 coded:

Standard Trap:

Trap Item	Meaning
"snmp-Trap"	trap community
.1.3.6.1.4.1.272	enterprise OID (=.iso.org.dod.internet.private.enterprise.bintec)
192.1.2.3	IP address
linkUp	trap type (coldStart, warmStart, linkDown, linkUp, authentificationFailure)

Trap Item	Meaning
0	no meaning
0:33:58	time stamp
"BIANCA/BRICK-XL"	system description
"brick"	system name
ifoperstatus.10001.4 = up	interface state

Enterprise-Specific Trap:

Trap Item	Meaning
"snmp-Trap"	trap community
.1.3.6.1.4.1.272	enterprise OID (=.iso.org.dod.internet.private.enterprise.bintec)
192.1.2.3	IP address
enterprise specific	trap type
row identifier (integer)	table number (=table number * 1000 + row number)
0:33:58	time stamp
"BIANCA/BRICK-XL"	system description
"brick"	system name
isdnchState.2000.1.1 = connected	trap variable

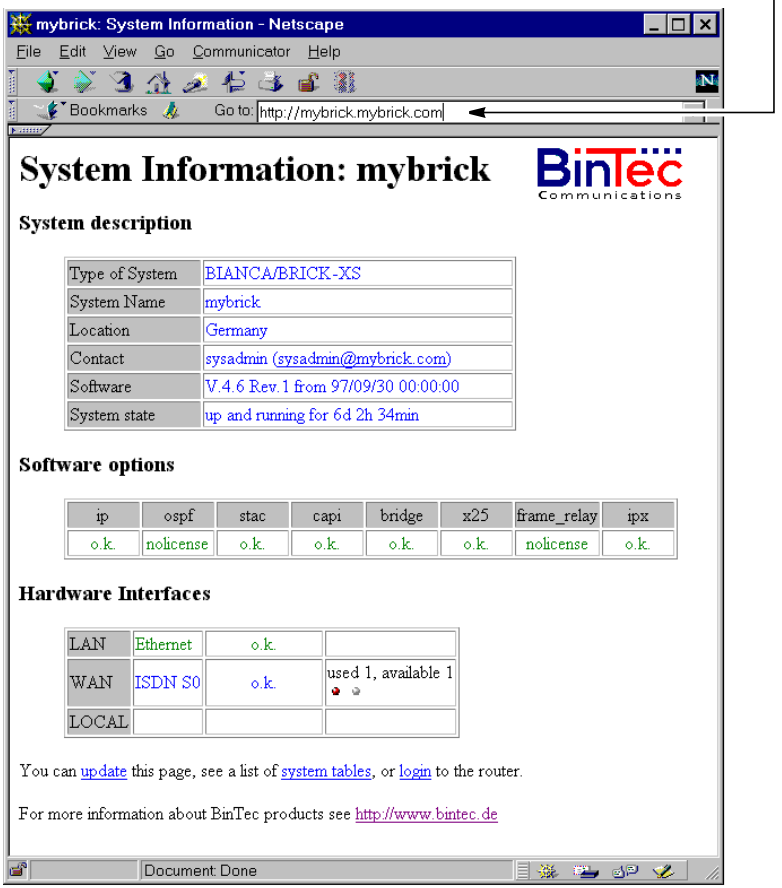
Web Based Monitoring

The BRICK's operational state can be quickly polled via an HTTP server that has been implemented on the BRICK. This server provides a status page which can be accessed from any WWW browser that supports HTML tables and the HTML 2.0 standard (i.e.,

Netscape's Mozilla or Microsoft Internet Explorer). The status page displays general system information, which licenses are installed, and current activity for each LAN or WAN interface.

Simply point a WWW browser at the BRICK using the following URL. The http port is only required if it was changed from its default value of 80 .

http://<*System Name*>**<:**<*HTTP Port Number*>



SNMP-Table Browsing

The contents of the BRICK's SNMP tables can be browsed via HTTP browsers using the "system tables" link from the main Status-Page. Initially this link displays a list of all system tables found on the BRICK. From there, individual system tables can be selected; the BRICK creates the appropriate HTML pages on-the-fly showing the current contents of the respecitive variables.

The CGI (Common Gateway Interface) programs htmlshow and snmpquery, are also available on the BRICK and can be used to selectively display the values of one or more SNMP table objects.

CGI Program: htmlshow

The contents of one or more BRICK SNMP variables can be selectively displayed to any WWW browser using the htmlshow program.

Note:

Only the "http" user may access the htmlshow program. The BRICK authenticates htmlshow queries once per browser session by prompting the requestor for the http user's password. This value is definedin the ***biboAdmHttpPassword*** field of ***bintecsec***.

The basic syntax for using htmlshow is as as follows. Possible options are described below.

separates CGI program name from parameters

`http://`<*SysName*>**`/htmlshow?`**<*option=val*>**`&`**<*option=val*>

separates parameter strings

htmlshow Options:

`oid=`*snmp_oid*
This option is mandatory and specifies an SNMP object identifier (OID) to display. *snmp_oid* is not case-sensitive. An OID may be specified in one of the following ways:

1.A symbolic object identifier name, i.e.

```
.iso.org.dod.internet.mgmt.mib-2.interfaces.ifEntry.ifTable
```

2.An numerical object identifier, i.e.

```
.1.3.6.1.2.1.2.2.1
```

3.A unique MIB-2 or BinTec MIB table or variable name, i.e.

```
iftable
```

Object identifiers starting with a period (“.”) are taken to be absolute object identifiers; otherwise a relative object identifier is assumed. Relative object identifiers are searched for relative to MIB-2, i.e. .iso.org.dod.internet.mgmt.mib-2 or .1.3.6.1.2.1.

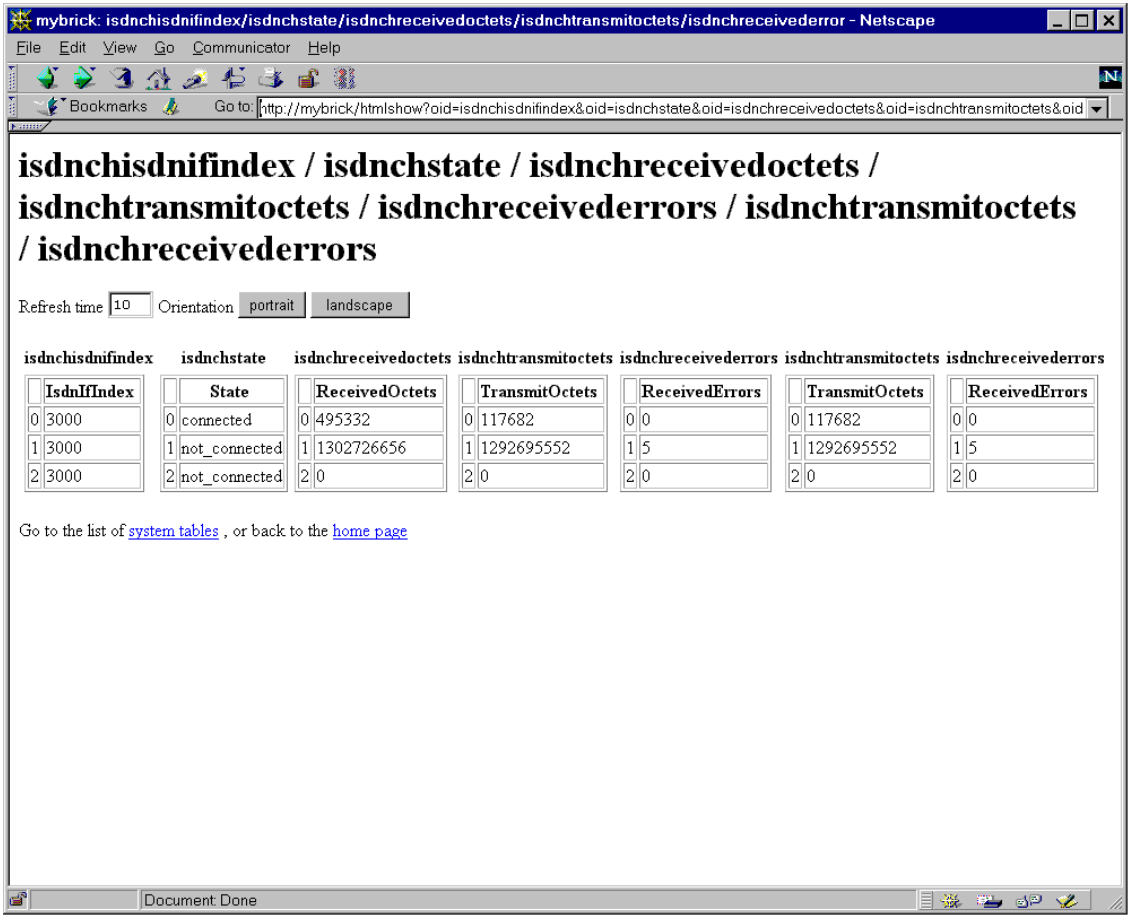
refreshtime=*interval*
If interval is specified the display is updated every *interval* seconds. Entering 0 in the resulting text field disables automatic refresh updates.

orientation=*mode*
Defines the orientation of the output. “portrait” (default) or “landscape” mode may be specified.

If more than one object identifier is specified, the resulting tables or columns are printed side-by-side. The following URL was used to display the selected system variables shown on the following page:

```
http://mybrick/htmlshow?oid=isdnChIsdnIfIndex&
oid=isdnChState&oid=isdnChReceivedOctets&
oid=isdnChTransmitOctets&oid=isdnChReceivedErrors&
refreshtime=10
```

TIP: References to HTML pages generated by the BRICK’s htmlshow program can be “bookmarked” for future reference. This will spare you the time of having to type long htmlshow queries (with the exception of the http password, all htmlshow options are saved in the bookmark)


mybrick: isdnchisdnifindex/isdnchstate/isdnchreceivedoctets/isdnchtransmitoctets/isdnchreceivederror - Netscape
File Edit View Go Communicator Help
Bookmarks Go to: http://mybrick/htmlshow?oid=isdnchisdnifindex&oid=isdnchstate&oid=isdnchreceivedoctets&oid=isdnchtransmitoctets&oid
isdnchisdnifindex / isdnchstate / isdnchreceivedoctets / isdnchtransmitoctets / isdnchreceivederrors / isdnchtransmitoctets / isdnchreceivederrors
Refresh time 10 Orientation portrait landscape
isdnchisdnifindex
IsdnIfIndex
0 3000
1 3000
2 3000
isdnchstate
State
0 connected
1 not_connected
2 not_connected
isdnchreceivedoctets
ReceivedOctets
0 495332
1 1302726656
2 0
isdnchtransmitoctets
TransmitOctets
0 117682
1 1292695552
2 0
isdnchreceivederrors
ReceivedErrors
0 0
1 5
2 0
isdnchtransmitoctets
TransmitOctets
0 117682
1 1292695552
2 0
isdnchreceivederrors
ReceivedErrors
0 0
1 5
2 0
Go to the list of system tables , or back to the home page
Document: Done

CGI Program: snmpquery

The contents of one or more selective SNMP object can also be polled from the BRICK using the snmpquery program. This program is similar to the htmlshow program but it does not format its output as HTML tables. (The output can still be read in any browser window). snmpquery is primarily intended for developers writing applications needing to access the BRICK's SNMP tables via the network.

The syntax for snmpquery is shown below. Exactly one oid=<value> parameter must be present within each HTML request. .

separates CGI program name from parameter

http://<*SysName*>**/snmpquery?oid=**<*value*>

Specifying Object Identifiers:

oid=*value*
An SNMP OID (object identifier) can be specified using an absolute name or a short-name (the same names available from the SNMP shell). Values beginning with a dot, ".", are assumed to be absolute names. Values not beginning with a dot are assumed to be relative to MIB-2.

Additionally, objects can be specified in numerical or symbolic format (alphabetical characters uppercase, lowercase, or mixed). For example, any of the following oid=<value> parameters shown below could be used to retrieve the contents of the tcp static table.

oid=.iso.org.dod.internet.mgmt.mib-2.tcp
(absolute name – symbolic format)

oid=.1.3.6.1.2.1.6
(absolute name – numeric format)

oid=tcp
(relative name – symbolic format)

oid=6
(relative name – numeric format)

snmpquery Output

The output of the snmpquery program consists of a header line followed by the contents of each requested SNMP object.

The header line consists of a numeric HTTP result code and a status message. The following result codes are currently defined.

Result Code	Status Message
200	OK
400	Bad Request
401	Unauthorized
404	Not Found
500	Internal Server Error

These codes are described in detail in RFC 1945 (*HTTP 1.0*).

SNMP variable information is then displayed. Each line consists of three columns:

1. The object identifier (absolute name – numeric format) enclosed in quotation marks.
2. The SNMP variable type.
3. The variable's current value. (DisplayString objects are also displayed in quotation marks).

A HTML request for the system table would be displayed as follows:

```
200 OK
".1.3.6.1.2.1.1.1.0"    DisplayString       "BIANCA/BRICK-XM"
".1.3.6.1.2.1.1.2.0"    ObjectIdentifier
".1.3.6.1.2.1.1.3.0"    TimeTicks           23924186
".1.3.6.1.2.1.1.4.0"    DisplayString       "J.D.Smith (smith@sample.com)"
".1.3.6.1.2.1.1.5.0"    DisplayString       "mybrick"
".1.3.6.1.2.1.1.6.0"    DisplayString       "John's desktop"
".1.3.6.1.2.1.1.7.0"    Integer             12
```

User Accounts

You can log into the BRICK using one of three different user IDs.

`Admin` **`Read`** **`Write`**

Passwords

For each user a separate password should be defined in the ***bintecsec*** table. Password information should be controlled. Default passwords (those set when your BRICK arrives) are shown below.

Object Name	USER ID	Password
biboAdmAdminCommunity	admin	bintec
biboAdmReadCommunity	read	public
biboAdmWriteCommunity	write	public

The password (value of the respective Community object in ***bintecsec***) defines the SNMP community name associated with all SNMP commands performed from the SNMP shell session.

User Rights

Each of the ***bintecsec*** users have a different level of access to the BRICK's configuration information. As the system administrator you will almost alway need to login as the admin user. The write and read users can be used to allow different levels of access to your system.

USER	Permission			
	System Table Editing	External System Commands	bintecsec Access	Setup Tool Access
admin	Read-Write	Execute	Read-Write	Execute
write	Read-Write	—	—	—
read	Read only	—	—	—

Other Passwords

HTTP Password

In addition to the SNMP community user passwords the bintecsec table contains the HTTP password for access to the BRICK's main Status page.

By default the ***biboAdmHttpPassword*** object is set to **`bintec`**.

Note The default HTTP password should be changed since it allows unrestricted read-access to all SNMP system tables on the BRICK via HTTP.

RADIUS Secret

The RADIUS secret used by the BRICK when contacting a configured RADIUS server (***biboAdmRadiusServer***) is also contained in ***bintecsec***.

By default the ***biboAdmRadiusSecret*** is left empty.

System Software Updates

The BRICK's system software is stored in flash RAM meaning that it can be easily updated allowing you to take advantage of newly developed/enhanced features not available when you purchased your BRICK.

BRICK system software updates are available via HTTP and FTP and are provided free of charge. You can always find the most recent software image for your BRICK via our WWW server at: http://www.bintec.de For sites limited to character based connections software images are also available via our FTP site at: ftp.bintec.de.

What's Needed

To update the BRICK's system software you will need the following:

- A BRICK system software image,
- A direct serial port connection between your BRICK and a PC where the software image is stored, –OR–
- An accessible (via a LAN or WAN interface) TFTP Server where the software image can be retieved from.

Performing a System Software Update

Software Update via TFTP

1. Retrieve the system software image you wish to install using one of the URLs (FTP/HTTP) mentioned above.
2. Place the software image in the TFTP server's TFTP directory. Normally[1], this is : **`C:\BRICK`** for PCs running the *DIME Tools TFTP Server* application or **`/tftpboot`** on UNIX workstations.
3. For UNIX TFTP servers ensure that the image is world-readable.
4. Log into your BRICK and issue the following command using the IP Address of your TFTP server and the image's filename.

   `update` *IP_Address filename*
5. Enter **y** (yes) when asked: **perform update (y or n) ?**
6. Enter **y** (yes) when asked: **Reboot now (y or n) ?**

Software Update via XMODEM

1. Retrieve the system software image you wish to install using one of the URLs (FTP/HTTP) mentioned above.
2. Place the software image on the PC your BRICK's serial port is connected to. Preferably *BRICKware for Windows* should also be installed on this system.
3. Start the **BRICK at COM** terminal program for the serial port the BRICK is attached to.
4. Now power up the BRICK (or reboot the system using the **`cmd=reboot`** command if it's already running).
5. At the BOOTmonitor prompt press the spacebar to activate the BOOTmonitor.
6. Select menu item (**3**) and simply answer the questions as prompted on the screen. You'll need to specify the location of the software image and begin the file transfer.
7. Once transferred you're given the option to update flash or write the image to memory. Select **u** update and then **b** boot the system.

1. The shown values are the defaults for most UNIX or PC systems, check your local configuration files to verify this location.

BOOT Options on the BRICK

When the BRICK boots up, it performs several self tests. When the tests are finished the BRICK optionally broadcasts BootP Requests via the first LAN interface if the IP address (for this interface is not configured).

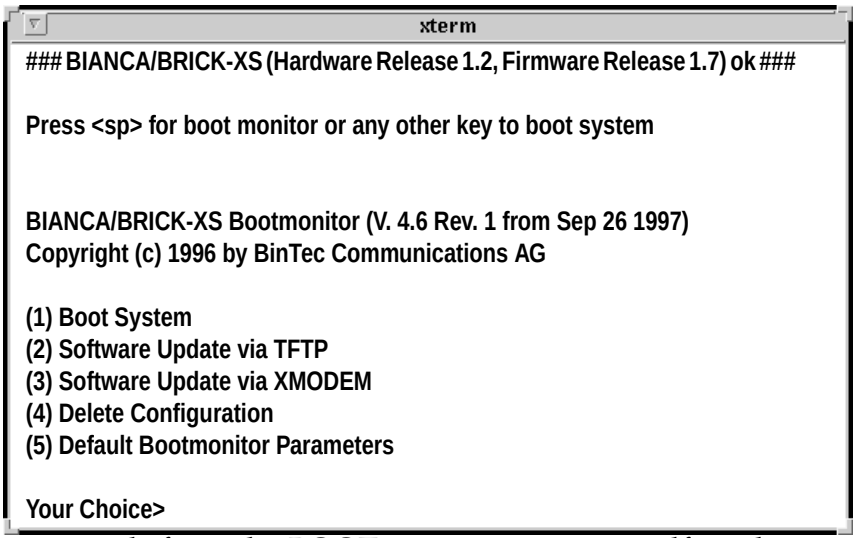
The BOOTmonitor

After the tests has been successfully completed, the BRICK switches into BOOTmonitor mode and displays a prompt to the screen.

Note that the BOOTmonitor is only displayed on terminals connected. directly to the BRICK's serial port. You will not see the BOTmonitor if connected via a LAN or WAN connection.

With the BOOTmonitor, you can easily perform firmware upgrades, test a new software release, or remove configuration files on your system.

To activate the BOOTmonitor the spacebar must be pressed within the first 4 seconds, otherwise the system continues with its normal boot procedure and switches into normal operation mode. Pressing the spacebar activates the BOOTmonitor as shown below. As long as the BOOTmonitor is active (or awaiting keyboard input), all front panel LEDs will remain on.



The commands from the BOOTmonitor menu are self guiding, informing/prompting you for confirmation along the way.

(1) Boot System

Select menu item (1) to load the compressed boot image from Flash into memory. This is the normal procedure performed at boot time.

(2) Software Update via TFTP

To upgrade the BRICK's system software via a TFTP server select option (2). You will be prompted for the following pieces of information:

- IP Address of an accessible TFTP Server (where the image is stored).
- IP Address of BRICK
- The file name of the software image to retrieve.

Once you've entered the information and the image has been successfully retrieved you will be asked to confirm the update. Here, you have two options:

(1.)Update Flash ROM
(2.)Write image to RAM and boot it.

Note

Note that option (2) only loads the image into RAM and does not remove your existing boot image stored in Flash. With this option, you can test the new software release without removing your existing boot image. If the BRICK is turned off, your old software release will be used upon a subsequent reboot.

(3) Software Update via XMODEM

You can upgrade system software via XMODEM over a serial connection with the BRICK by selecting this option. You will be prompted to verify the baud rate to use over the serial connection. The time required to transfer the file will depend on the size of the file and baud rate you've chosen.

As when performing an update via TFTP you will then be prompted to confirm the update as follows:

(1.)Update Flash ROM
(2.)Write image to RAM and boot it.

(4) Delete Configuration

Select option (4) to return the BRICK to its factory settings, as it arrived. All configuration files and BOOTmonitor parameters (see below) are removed.

(5) Default BOOTmonitor Parameters

Select option (5) from the menu to change the default settings used by the BOOTmonitor. These settings include:

- The baud rate used for serial connections.
- The LAN interface to use for TFTP file transfers.
- The Local IP address for the BRICK.
- The IP address for the TFTP server.
- The system software image file to download.
- Automatic boot file retrieval over TFTP

The IP address settings defined here are used strictly for the BOOTmonitor and are not used for any IP routing functions on the BRICK.

Note

If you change the baud rate, be sure that your terminal supports this rate, otherwise you may not be able to connect to the BRICK. The default setting is set at 9600 baud, which is supported by practically all terminals.

Automatic booting over TFTP

The BRICK can load its boot file via TFTP automatically at boot time by defining the appropriate settings in menu item (5). After setting the local and remote IP addresses, and the name of the system software image file to retrieve answer “yes” when asked the question:

```
Do you want to boot automatically from the TFTP server (y or n):
```

Booting via BootP

The BRICK's initial configuration information can be loaded remotely using a BootP server on the local network. This initial information normally includes the BRICK's IP address and the name of its configuration file but may include other information. The server that provides this information may be a UNIX workstation running a bootpd process (see: Setting Up a BootP Server) or a PC running the included *DIME Tools BootP Server* program (see: BRICKware for Windows).

During every system startup, the BRICK starts a BootP client process. Until an IP address is assigned, this process broadcasts standard BootP REQUEST packets every five seconds over the local network. Depending on how your BootP server is configured, the BRICK can also load it's configuration file remotely using TFTP. As soon as the IP address is received, the bootpd (client) process is ended.

Various information can be transmitted to the BRICK using a BootP server. The BRICK BootP client process accepts the following BootP information (or TAGs) in accordance with the following Request For Comments (RFCs).

	TAG	RFC
Subnet Mask	1	1048
TimeServer	4	1048
TimeOffset	2	1048
IP Address	-	951
Host Name1	2	1048
Domain Name	15	1395
Domain Name Server	6	1048
Log Server	7	1048
TFTP Bootfile	-	951

Note

If the BootP server sends a hostname, domain name, or name server information, the BRICK will accept this information (by setting the respective variables) only if this information hasn't already been set.

BootP Relay Agent

The BRICK can also serve as a BootP Relay Agent for other hosts on the LAN. This is useful for stations that need to retrieve boot information remotely from a BootP server, but aren't on the same physical IP network as the server. If the BRICK is on the same IP network as the station, it receives the stations BootP requests, and forwards them to server defined in *biboAdmBootpRelayServer*. See the section BootP Relay Agent Settings under BOOTP and DHCP in Chapter 7.

Other System Administration Tasks

Setting Up a BootP Server

To configure a BootP server on a UNIX workstation follow these instructions. The information shown below briefly describes setting up BootP to provide the BRICK with basic IP settings (IP address, netmask, and name server's address). Refer to your local documentation for detailed description for your specific platform.

1. Edit (or create) the */etc/bootptab* file to include the following lines:

```
brick:\
    :ht=<the Hardware Type is usually "ether">:\
    :ha=<the BRICK's Hardware (or MAC) Address>:\
    :ip=<the IP Address to use>:\
    :sm=<the Subnet Mask to use>:\
    :ds=<the Domain Name Server's IP Address>:
```

Note

The very first tag identifies the hostname this bootptab entry applies to. By default this is "brick" on systems where ***sysName*** hasn't been configured. If the system name is already configured specify that value here

2. You can start the bootpd process from the command line using:

   On Solaris 2.5 and SunOS Systems:

```
/etc/bootpd -s
```

   On Linux Systems:

```
/usr/sbin/bootpd -s
```

3. You may want to start the bootp daemon from the Internet Services daemon by adding the appropriate line to the */etc/inetd.conf* file:

   On Solaris 2.5 and SunOS Systems:

```
bootps dgram udp wait root /etc/bootpd bootpd
```

   On Linux Systems:

```
bootps dgram udp wait root /usr/sbin/bootpd \
bootpd bootptab
```

4. If you've added the bootps entry to /etc/inetd.conf as in step 3 you'll have to restart the inetd process for your changes to become effective.

   On Solaris 2.5 Systems,
   ps -ef |grep inetd
   kill -1 <*pid*>

   On SunOS and Linux Systems,
   ps -ax |grep inetd
   kill -1 <*pid*>

   where <**pid**> is the process id of your running inetd process.

Setting up a TFTP Server

The TFTP (Trivial File Transfer Protocol) allows configuration files to be transferred to/from remote machines. The BRICK implements TFTP allowing you to send and receive files to/from hosts where a TFTP server is running. The TFTP server may be a UNIX host or a PC running DIME Tools' TFTP Server application (see: BRICKware for Windows). A brief description of setting up a TFTP server on a UNIX workstation is covered below.

1. Allow the TFTP daemon to start. This is normally done by inserting one of the lines shown below in your */etc/inetd.conf* file. Normally the correct entry is already present in the file and all you have to do is uncomment it. Refer to your local documentation inetd and tftpd) for more specific instructions.

   On Solaris 2.5:
   ```
   tftp dgram udp wait root /usr/sbin/in.tftpd \
   in.tftpd -s /tftpboot
   ```
   On SunOS Systems:
   ```
   tftp dgram udp wait root /usr/etc/in.tftpd \
   in.tftpd -s /tftpbooot
   ```
   On Linux Systems:
   ```
   tftp dgram udp wait nobody /usr/sbin/tcpd \
   in.tftpd /tftpboot
   ```

2. Create the TFTP directory. You must separately create the TFTP directory (last field of the TFTP entry in *inetd.conf* shown above) and make it world readable using:

```
mkdir /tftpboot
chmod 777 /tftpboot
```

3. Restart the inetd process. After you have added the above line to your local /*etc/inetd.conf* file you must restart the inetd process. You must determine the process ID of inet daemon and restart the process. You can use the standard ps and kill commands as follows:

   On Solaris 2.5 Systems

```
ps -ef |grep inetd
kill -1 <pid>
```

   On SunOS or Linux Systems:

```
ps -ax |grep inetd
kill -1 <pid>
```

   where <**pid**> is the process id of your running inetd process.

Remember that before you send TFTP files from the BRICK to your (UNIX) TFTP server you must create the destination file in the TFTP directory and it must be world readable. This could done using the commands:

```
touch /tftpboot/brick.cf
chmod 777 /tftpboot/brick.cf
```

Special Note:

TFTP Servers

Some UNIX TFTP server implementations (in particular ***older BSD based systems***) do not reset the file length to 0 bytes prior to writing the TFTP file in response to a TFTP Write-Request; i.e., cmd=put or cmd=state is used.

This results in leftover data at the end of the TFTP file after the new data has been written. These files can not be processed by the BRICK.

Setting Up a syslog Daemon

Log hosts configured on the BRICK can be a PC running *DIME Tools Syslog Daemon* program or a UNIX workstation running a syslog daemon. This section briefly explains setting up an */etc/syslog.conf* file for a UNIX worstation.

The exact format of this configuration file may be different on your UNIX platform, see your local documentation for more specific information.

1. As root edit the **`/etc/syslog.conf`** file to include the appropriate logging entry (see below). A typical logging entry that would save messages to a predefined file might look like this.

```
#facility.level     action
local0.info         /var/adm/brick.log
```

2. For actions that specify a log file, make sure you create the file and it has read-write permission for the syslog daemon.
3. Then as root stop and restart the syslog daemon.

   On Solaris Systems

```
/etc/init.d/syslog start
/etc/init.d/syslog stop
```

   On SunOS Systems

```
kill -1 ‘cat /etc/syslog.pid‘
```

   On Linux Systems:

```
kill -1 ‘cat /var/run/syslogd.pid‘
```

4. If you haven’s already done so configure this host as a log host on the BRICK. (See:).

Logging Entries in /etc/syslog.conf

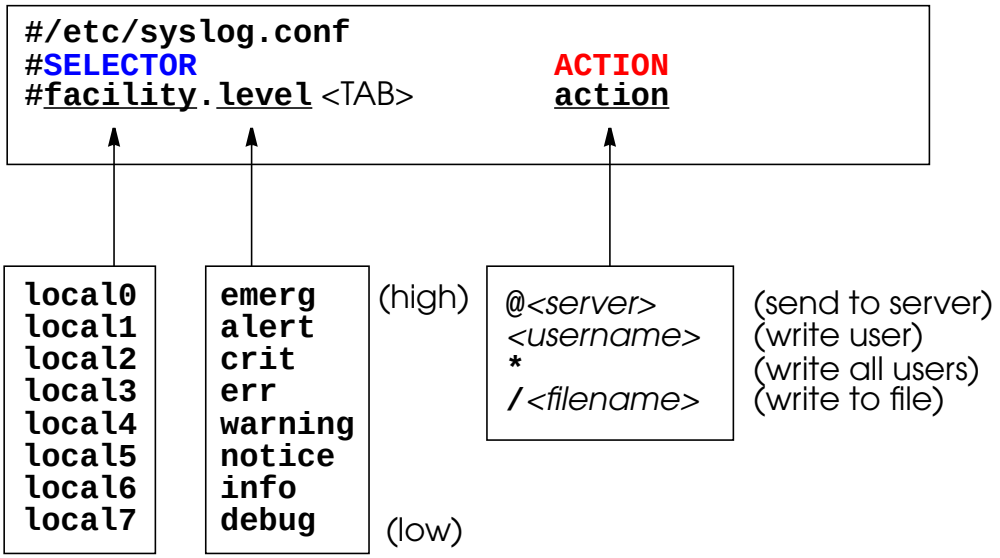
Logging entries in this file consists of two TAB-sparated fields referred to as:

SELECTOR and **ACTION**

The **SELECTOR** field consists of a **facility.level** pair separated by a dot. (Actually, selector can contain multiple facility.level pairs separated by semi-colons, however, for the sake of simplicity we'll asusme only one pair is being used.). The facility part identifies a system faciltiy that a system message is received over; a sort of incoming port number if you will. The level part identifies the severity associated with the message.

The **ACTION** field identifes the action to take upon receiving a system message via this facility. Actions might include saving the tsystem message to a file, writing to a specific user (if currently logged in), or forwarding the message to the syslogd of another host.

The facility and level of an incoming system message (i.e., sent from the BRICK) must match both facility and level before the syslog daemon performs the action. The values that may be used in these field when configuring logging entries for the BRICK are as follows:



Note:

On most systems the facility field must match the facility of the transmitting host or be "*".
On most systems a level entry or X will match All messages (arriving on the respective facility) with levels ≥ X.
Some systems (Linux) support additional extensions in the level field to match level subsets.

Setting up a Time Server

The BRICK acts as a Time Client and needs a Time Server to retrieve the time from. There are various possibilities: time can be retrieved from ISDN; the Time Server protocol via "Time Service UDP" is available on the Windows software package, BRICKware; the time protocols "Time Service UDP/TCP are usually available on all Unix hosts; an XNTP Server package is freely available for PC/Unix servers, enabling the SNTP protocol via UDP.

Depending on the kind of server used, the BRICK can retrieve the current time using any of the following four methods:

- Time Service (RFC 868) via UDP
- Time Service (RFC 868) via TCP
- Simple Network Time Protocol (SNTP) (RFC 1769)
  Via individual Time Requests or Broadcasts: in the latter case, no explicit time requests are necessary, the Time Server automatically sends network broadcasts to all its time clients at regular intervals, thus saving packet traffic.
- ISDN D-channel (stack 0 only)

The following relevant SNMP variables are configured on the BRICK in the *Admin* system table:

biboAdmTimeServer Specifies the IP-address of the Time Server in dot-format

biboAdmTimeOffset Specifies the time in seconds to add/subtract to the retrieved time. Values between -24 and +24 are assumed to be hours and are appropriately converted to seconds. Note that when time is retrieved from ISDN the offset must be set to zero.

biboAdmTimeProtocol Specifies the protocol to use to retrieve current time. Regarding the four methods noted above, the following protocols are possible.

- time_udp: Time Service (RFC 868) via UDP
- time_tcp: Time Service (RFC 868) via TCP
- time_sntp: SNTP (RFC 1769) via UDP
- isdn: ISDN D-Channel (stack 0 only)

- none: Disable time retrieval altogether

biboAdmTimeUpdate Specifies the interval in seconds at which current time should be updated/retrieved. As with Time Offset values between -24 and +24 are assumed to be hours and converted to seconds. For Protocol=time_udp, time_tcp, or time_sntp (if not in Broadcast mode) new requests are sent every ***biboAdmTimeUpdate*** seconds. When isdn is used, the current time is retrieved from the next ISDN connection established after ***biboAdmTimeUpdate*** seconds.

6

Chapter Six

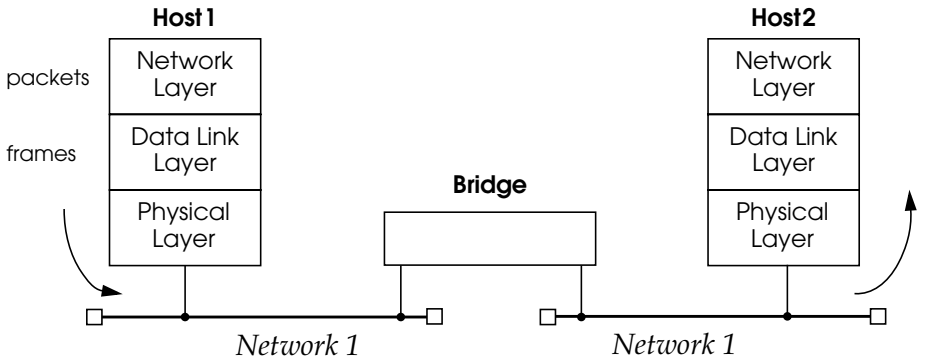
CONFIGURING THE BRICK AS A BRIDGE

What's Covered?

- Background on Bridging
- Bridging with the BRICK
  - Bridging Features
    - Learning Bridges
    - The Spanning Tree Algorithm
    - Bridge Filtering
- Configuring Bridging on the BRICK
  - Enabling Bridging
  - Bridge Initialization
- Using the BRICK as a Bridge
  - Bridging between LANs
  - Bridging over WAN Links
  - Controlling Bridging Activity Using Filters

Background on Bridging

Bridging is one of the easiest ways to connect network segments. A bridge is attached to two or more networks and simply forwards frames between them. The contents of these frames are of no concern to the bridge; frames are forwarded unchanged.



In transparent bridging each bridge makes its own routing decisions and is therefore 'transparent' to the communicating hosts on the end networks. Additionally, a transparent bridge configures itself (in terms of routing information) after coming into service.

Because a bridge forwards complete frames between connected networks many different protocols can coexist on either network, the messages are forwarded unchanged (protocol information is passed as raw data in the ethernet frames). Bridges are used when multiple-protocol packets need to be shared among networks.

Bridging with the BRICK

Bridging Features

As with most bridges you simply have to connect the BRICK to two or more network segments and turn bridging on. However, several bridging features are available on the BRICK that can be used to overcome some of the limitations inherent in bridging.

Learning Bridges

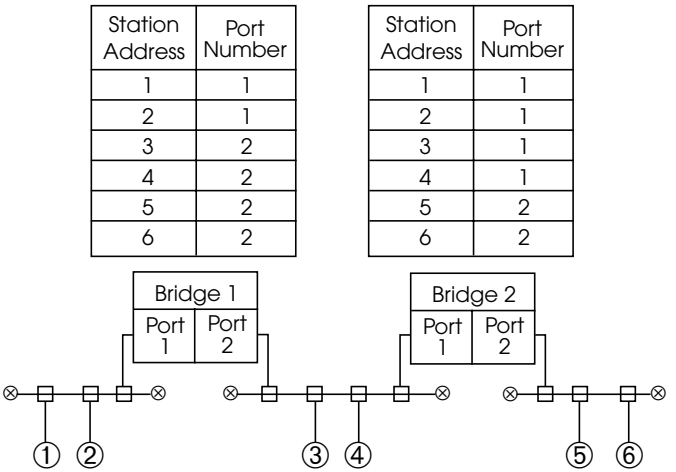
The BRICK is a learning bridge. Using the source and destination hardware addresses the BRICK decides which physical interface to forward each packet it receives. This decision is made by consulting its forwarding database, the ***dot1dTpFdbTable***. Each entry in this table has the following fields:

Address	Contains MAC (Medium access control) addresses.
DestPortIfIndex	The respective BRICK interface number to use when bridging frames for this address.
Status	How this information was learned.
Age	How old this information is (see below).

When the BRICK first comes into service, its forwarding database is empty. Then, when a frame is received, the source address of the frame and the interface it was received on are entered into the database. Since the destination interface is not yet known, a copy of the frame is broadcasted on each of the BRICK's interfaces. As frames are propagated, other learning bridges perform the same procedure. This allows other bridges on the network to rapidly build up their forwarding database.

To ensure that the ***dot1dTpFdbTable*** is current, and doesn't get too large, each entry has an ***Age*** associated with it. Whenever a frame is received from that address this field is reset. If no frame arrives before the Aging Timer expires, the entry is removed from the database. The BRICK uses the ***dot1dTpAgingTime*** variable which is set to 300 seconds by default.

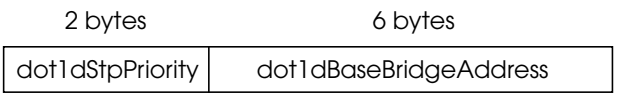
A simple bridge example is shown below.



The Spanning Tree Algorithm

The simple learning procedure explained above is only sufficient for simple networks; i.e. multiple paths between two segments can not exist since the learning process would constantly cause entries to be overwritten. In such cases the BRICK uses an additional mechanism, known as the Spanning Tree Algorithm that compensates for network topologies with multiple paths between stations.

The spanning tree algorithm defines special frames (messages) known as bridge protocol units (BPDU) which are exchanged among all bridges on a network. Each bridge is uniquely identified by an 8 byte Bridge ID. On the BRICK this ID is determined using the ***dot1dStpPriority*** and ***dot1dBaseBridgeAddress*** objects as follows:



Also one bridge must be singled out as the root bridge; this is the bridge with the smallest identifier and the highest priority value. After the root bridge has been chosen, each bridge determines which port offers the lowest cost path to the root (its root port). The root bridge's address is stored in the ***dot1dStpRootPort*** object. Configura-

tion BPDUs are transmitted at regular intervals, defined by the Hello Time (***dot1dStpHelloTime***), to ensure that the root bridge information is current.

Bridge Filtering

Bridge Filtering allows you to control the amount of traffic that may be passed over the BRICK's interfaces. This is an important tool when bridging over ISDN links since every WAN connection costs money. Bridge Filters follow the same concept used with Access Lists in IP Routing consisting of Allow and Deny Tables as follows.

dot1dStaticAllowTable	Defines packets that may be bridged.
dot1dStaticDenyTable	Defines packets that may NOT be bridged.

Each table consists of the following fields:

SrcIfIndex	The BRICK interface the frame was received on.
DstIfIndex	The BRICK interface the frame would be forwarded on.
ByteOffset	The number of bytes to skip (starting from the beginning) before making a comparison.
Mask	Which bytes are compared.
Value	The actual value to look for in the packet.
Status	The status of this entry.
Age	The age of this entry in seconds (used with the ***Status*** field).

Using these fields you can filter packets based on one or more criteria:

1. The packet's Source Interface
2. The packet's Destination Interface
3. A specific field in the Ethernet Frame

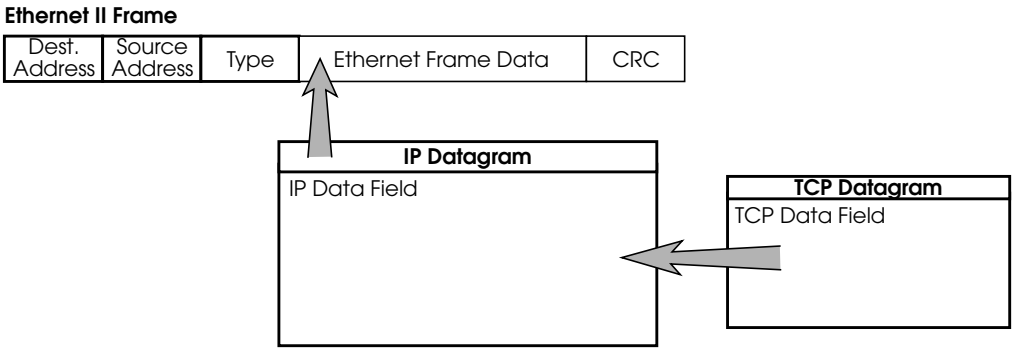
Filter Matching Procedure

Bridge filtering is sometimes referred to as packet-filtering because the decision to allow or deny (filter) a packet is based on the contents of the packet. As frames are received the contents are compared to each defined filter, starting with the Allow Table, then the Deny Table. A packet is said to match a filter if all its conditions are met. A filter condition may be one of the following.

SrcIfIndex = the interface this frame was received on.
DstIfIndex = the interface this frame would be forwarded on.
Value = the contents of the frame starting from ***ByteOffset*** bytes

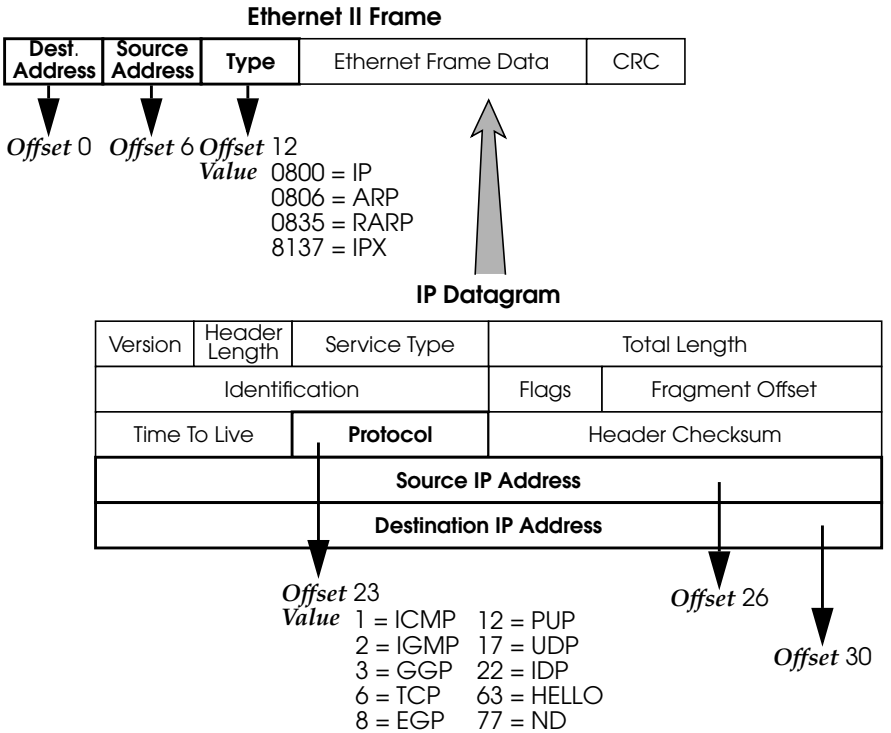
NOTE: A "0" in the ***SrcIfIndex*** and/or ***DstIfIndex*** fields means match any interface.

The common practice of encapsulating datagrams within datagrams (as shown below) poses no problem to the BRICK. As a bridge it doesn't differentiate the individual fields, it simply sees the complete packet as a sequence of bytes. Using the ***ByteOffset*** and ***Value*** fields you can define filters that are based on the actual contents of the frame.

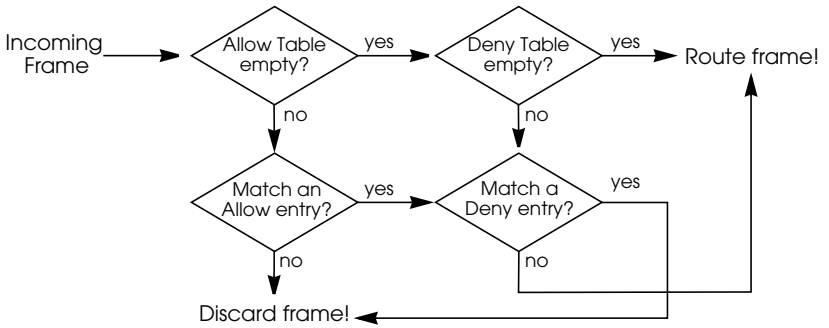


When filtering packets based on contents of the ethernet frame it's important to know the relative locations (offset) and possible values of the frame's fields. Below is

an ethernet frame that highlights some of the more interesting fields (with respect to bridge filtering).



The packet is compared with all Allow and Deny Table entries and a decision is made based on the following algorithm.



Configuring Bridging on the BRICK

Enabling Bridging

There are two basic requirements to fulfil before the BRICK begins bridging.

1. The ***biboAdmBridgeEnable*** object must be set to "enabled".
2. At least two interfaces must be enabled in the ***dot1dStpPortTable.*** Here, two or more interfaces' ***Enable*** field must be set to "enabled".

NOTE: Both steps can be accomplished using Setup Tool. Refer to Chapter 5 of the User's Guide.

Bridge Initialization

Once bridging has been enabled, the system goes through three internal phases before bridging can actually take place.

1. **Listening**—During the listening phase the system transmits configuration BPDUs and evaluates any others it receives. In this phase the spanning tree is computed and the root bridge is determined. The bridge ID of the root bridge is then set in ***DesignatedRoot*** variable of the ***dot1dStpPortTable***.
2. **Learning—**The system then switches to the learning phase where all frames it receives are evaluated.

3. **Forwarding**—After a preset time (defined in IEEE802.1d-1990), the BRICK switches to the forwarding state.

Once the system reaches the forwarding state, the BRICK checks the ***dot1dStpPortTable*** to see which interfaces are available for bridging.

Using the BRICK as a Bridge

Below are some examples of setting up the BRICK as a bridge.

- In the first example we'll use the BRICK to bridge between two local LAN segments (i.e., a BRICK-XM or BRICK-XL with two LAN interfaces is assumed.)
- In the second example we'll use the BRICK as a bridge to connect a local LAN with a remote LAN over a dialup ISDN link.
- In the last example we'll extend the previous example showing you how to add filters to control bridging traffic and to save money.

Bridging between LANs

Step 1

First make sure the bridging service is enabled on the BRICK.
The ***biboAdmBridgeEnable*** variable must be set to "**enabled**".

```
mybrick: > biboAdmBridgeEnable=enabled

biboAdmBridgeEnable( rw):       enabled

mybrick : admin>
```

Step 2

Next we need to enable the interfaces we want to bridge between. The ***dot1dStpPortTable*** lists all available interfaces for bridging. For ethernet interfaces

you must use the "*-llc" interface. This ensures that the LLC (Link Layer Control) frame format is used.

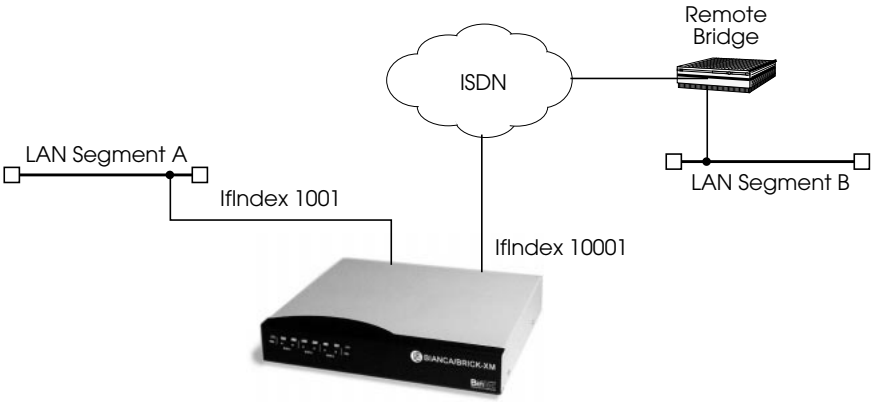
```
mybrick : admin> dot1dStpPortTable

inx IfIndex(*ro)                Number(ro)              Priority(rw)
    State(ro)                   Enable(rw)              PathCost(rw)
    DesignatedRoot(ro)          DesignatedCost(ro)      DesignatedBridge(ro)
    DesignatedPort(ro)          ForwardTransitions(ro)  BackupForIfIndex(rw)

 00 1001                        0                       0
                                disabled                0
    80:0:0:a0:f9:0:e:91         0                       80:0:0:a0:f9:0:e:91
    0                           0                       0

00  2001                        0                       0
                                disabled                0
    23:a0:21:a3:f5:0:d:88       0                       80:0:0:a0:f9:0:e:91
    0                           0                       0

mybrick : dot1dStpPortTable> IfIndex:00=enable IfIndex:01=enable
```

Step 3

That's all that is required for our setup. The BRICK will now run through it's Bridge Initialization functions. You can optionally see the state of the bridging interfaces by displaying the ***dot1dStpPortTable***. You can also see the list of learned bridging entries by displaying the ***dot1dTpFdpTable***.

```
mybrick : admin> dot1dTpFdbTable

inx Address(*rw)          DestPortIfIndex(rw)       Status(-rw)
    Age(ro)

    00 0:a0:f9:0:e:91     1001                      self
    0 00:12:47.00

    00 0:a0:f9:0:c:2c     2001                      self
    0 00:12:47.00

    00 0:a0:f9:0:b:12     1001                      learned
    0 00:12:47.00

mybrick : dot1dTpFdbTable >
```

Bridging over WAN Links



Step 1

We'll assume the bridging service has been enabled (see previous example). First, create a new PPP interface for the Remote Bridge. Creating PPP interfaces is covered in chapter 7. Here we set the ***Type*** field in the ***biboPPPTable*** to "**`isdn_dialup`**". The

BRICK will assign a unique numerical value to the ***IfIndex*** field that we'll need in Step 2.

```
mybrick: admin > biboPPPType=isdn_dialup
05: biboPPPType.1.5( rw):      isdn_dialup

mybrick : biboPPPTable > biboPPPTable
inx  IfIndex(ro)              Type(*rw)              Encapsulation(-rw)
     Keepalive(rw)            Timeout(rw)            Compression(rw)
     Authentication(rw)       AuthIdent(rw)          AuthSecret(rw)
     IpAddress(rw)            RetryTime(rw)          BlockTime(rw)
     MaxRetries(rw)           ShortHold(rw)          InitConn(rw)
     MaxConn(rw)              MinConn(rw)            Callback(rw)
     Layer1Protocol(rw)       LoginString(rw)

05  10006                     isdn_dialup            ppp
     off                      3000                   none
     none
     static                   4                      300
     5                        20                     1
     1                        1                      disabled
     data_64k

mybrick : biboPPPTable>
```

Step 2

Now we can add the Remote Bridge's telephone number to the ***biboDial-Table***. We set the ***IfIndex*** and the ***Number*** field in one operation. In the ***IfIndex*** field, we use the number assigned by the BRICK in the previous step.

```
mybrick: biboPPPTable > biboDialIfIndex=10006 biboDialNumber=555

06: biboDialIfIndex.10006.6( rw):      10006
06: biboDialNumber.10006.6( rw):       "555"

mybrick : biboDialTable >
```

Step 3

Now we can enable the local and the remote interfaces to bridge between. The ***dot1dStpPortTable*** should have an entry for our local ethernet segment as well as our new PPP partner interface.

```
mybrick : admin> dot1dStpPortTable

inx IfIndex(*ro)                Number(ro)              Priority(rw)
    State(ro)                   Enable(rw)              PathCost(rw)
    DesignatedRoot(ro)          DesignatedCost(ro)      DesignatedBridge(ro)
    DesignatedPort(ro)          ForwardTransitions(ro)  BackupForIfIndex(rw)

 00 1001                        0                       0
    forwarding                  disabled                0
    80:0:0:a0:f9:0:e:91         0                       80:0:0:a0:f9:0:e:91
    0                           0                       0

05  10001                       0                       0
     broken                     disabled                0
                                0
    0                           0                       0

mybrick : dot1dStpPortTable> IfIndex:00=enable IfIndex:05=enable
```

The configuration is complete. We can optionally verify that bridging has started by displaying the ***dot1dStpPortTable*** and the ***dot1dTpFdpTable*** as mentioned in the previous example.

NOTE: For most sites bridging over WAN links is less desirable due to the possibility of increased ISDN costs incurred through dialup connections. With careful consideration and planning however bridge filters can be used to make bridging over WAN links a viable alternative.

Further optional settings are afforded by the following variables, which give you additional influence over your WAN-link bridges:

Delay before Change of State

The first is ***dot1StpBridgePPPForwardDelay*** and is an addition to the ***dot1dStp*** table. The unit of the value of this variable is 1/100 seconds, the range lies between 100 and 3000 and the default value is 500 (= 5 seconds).

This variable defines how long the port of a PPP connection (leased line or dial-up connection) should wait before a change of state may take place. After the establishment of an ISDN connection, the state takes this set period of time to change from listening to learning, then the same time again to change from learning to forwarding. ***dot1StpBridgePPPForwardDelay*** only affects WAN connections like PPP and X.25.

When the default value (500) is set, it takes 5 seconds to change the state from listening to learning and another 5 seconds to change from learning to forwarding. After an ISDN connection has been made, it consequently takes 10 seconds until data is transmitted in the state forwarding.

This period of time is necessary to detect redundant paths.

Backup for a Leased Line

The second variable is ***dot1dStpPortBackupForIfIndex*** and is an addition to the ***dot1dStpPortTable***.

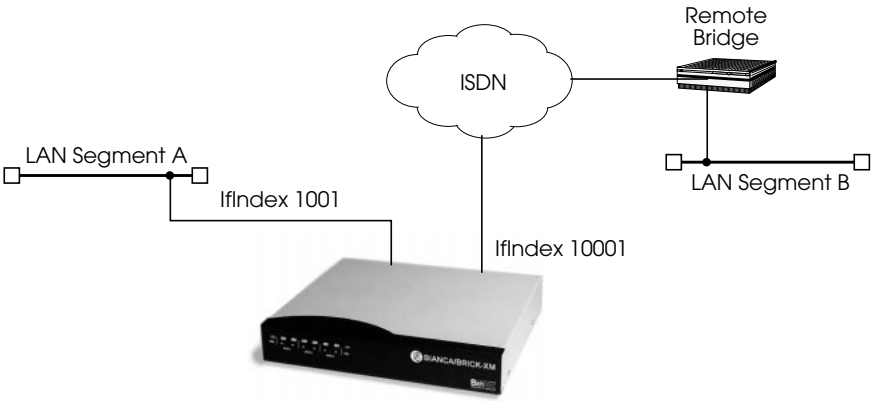
This variable is conceptualised for a situation in which two BRICKs are bridging two local networks over PPP connections. One of these is a leased line, the other a dialup line.

dot1dStpPortBackupForIfIndex is used to configure the dialup connection as a backup connection for the leased line connection.

In the ***dot1dStpPortTable***, set the value of the ***dot1StpPortBackupForIfIndex*** variable for the interface of the dialup line with the same value as in the ***dot1dStpPortIfIndex*** of the leased line port. This effectively makes the port of the dialup line serve as the backup link for the leased line. As long as the leased line functions properly (its state is forwarding), the dialup link is not established. Should the leased line fail, however, the dialup link (backup) is established and the entries for the port of the leased line in the ***dot1dTpFdbTable*** are deleted.

Controlling Bridging Activity Using Filters

Now we want to show you how to use Bridge Filters to control bridging traffic. Bridge filters are most commonly used when bridging over ISDN WAN links to minimize costs. Remember that bridge filtering is based on the contents of the ethernet frame. An overview of the ethernet frame format was covered here.



Following are three examples that could be used to extend the previous example for bridging over WAN links. We'll assume bridging is already configured so we can focus on the filter entries. The following three examples show how to:

1. **Filtering frames sent from a particular host (by MAC address)**
   Here we want to single out packets from a specific host by filtering the MAC address field of the ethernet frame.
2. **Filtering all IPX packets coming from the local LAN**
   Here we want to filter out all IPX packets, this can be done by filtering out the Type field in the MAC header.
3. **Filtering broadcast packets**
   Here we want to filter out all broadcast packets (destination address field is ff:ff:ff:ff:ff:ff in hexadecimal).

Filtering frames sent from a particular host (by MAC address)

To filter out all frames sent from a specific host we first need the host's MAC (hardware) address. Then, all we need is one Deny Filter that filters out all frames sent from this host.

Assuming our source host had a MAC address of 0:a0:f9:0:e:19 and was attached to the BRICK's first ethernet interface (en1 or 1001) our filter would be created as follows.

```
mybrick: admin > dot1dStaticDenyTable

inx SrcIfIndex(*rw)          DstIfIndex(*rw)          ByteOffset(rw)          Mask(rw)
    Value(rw)                Status(-rw)              Age(rw)

mybrick: dot1dStaticDenyTable >SrcIfIndex=1001 DstIfIndex=0 ByteOffset=6
                               Value=0:a0:f9:0:a0:19

00: dot1dStaticDenySrcIfIndex.1001.0( rw):          1001
00: dot1dStaticDenyDstIfIndex.1001.0( rw):          0
00: dot1dStaticDenyByteOffset.1001.0( rw):          6
00: dot1dStaticDenyValue.1001.0( rw):               0:a0:f9:0:a0:19

mybrick: dot1dStaticDenyTable > dot1dStaticDenyTable

00  1001                     10001                    6
    0:a0:f9:0:a0:19          permanent                0 00:03:24.00

mybrick: dot1dStaticDenyTable >
```

NOTE: Since the ***SrcIfIndex*** and ***DstIfIndex*** fields are index variables (required for creation of new table entries) we also specify them here. Also rememer that the special value "0" matches all interfaces.

Filtering all IPX packets coming from the local LAN

A single Deny filter is all that's needed to filter out all IPX packets originating on a specific LAN segment. IPX packets are identified by the protocol ID of 0x8137 in the Type field of the Ethernet frame. This filter will ensure that no IPX packets coming from the LAN segment are bridged to our ISDN interface at 10001.

As in the previous example we'll assume basic bridging has already been configured. Our filter would be created as follows.

```
mybrick: dot1dStaticDenyTable >

mybrick: dot1dStaticDenyTable > SrcIfIndex=1001 DstIfIndex=10001
                                ByteOffset=12 Value=81:37

01: dot1dStaticDenySrcIfIndex.1001.0.1( rw):          1001
01: dot1dStaticDenyDstIfIndex.1001.0.1( rw):          10001
01: dot1dStaticDenyByteOffset.1001.0.1( rw):          12
01: dot1dStaticDenyValue.1001.0.1( rw):               81:37

mybrick: dot1dStaticDenyTable > dot1dStaticDenyTable

inx SrcIfIndex(*rw)          DstIfIndex(*rw)          ByteOffset(rw)          Mask(rw)
    Value(rw)                Status(-rw)              Age(rw)

01  1001                     10001                    0
    81:37                    permanent                0 00:10:06.00

mybrick: dot1dStaticDenyTable>
```

NOTE: If we had several exceptions to this rule to account for (filter all IPX packets except those from hosts x,y, and z) we would create either a series of Allow entries or Deny entries for individual host MAC addresses depending on which required the least entries.

Filtering broadcast packets

Broadcast packets can also be easily filtered out using the "Destination Address" field in the MAC frame. Broadcast addresses can be identified by the value ff:ff:ff:ff:ff:ff

7

Chapter Seven

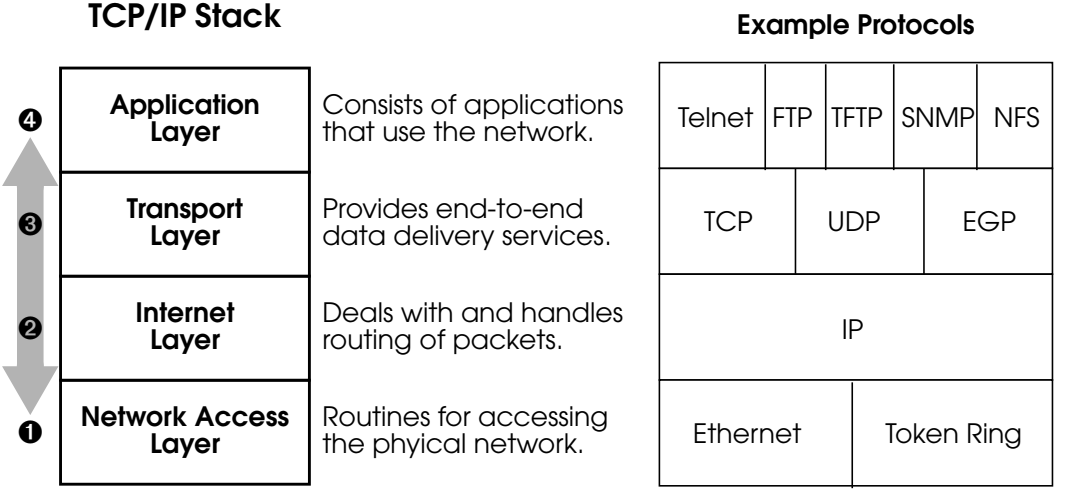
CONFIGURING THE BRICK AS AN IP ROUTER

What's Covered?

- TCP/IP Primer
  - Encapsulation
  - IP Addressing
  - Subnetting
  - Protocols, Ports and Sockets
- IP Routing Protocols
  - RIP
  - OSPF
  - The Point-to-Point Protocol
- DialUp IP Interfaces
  - Creating a DialUp IP Interface
  - DialUp Options
- Dual IP Address Interfaces
- IP Routing on the BRICK
- Extended IP Routing
- BOOTP and DHCP
  - BootP Relay Agent Settings
  - DHCP Server Setting
  - DNS and WINS (NBNS) Relay
- DNS and WINS Addresses over PPP
- Dynamic IP Address Assignment
  - Server Mode
  - Client Mode
- Routing with OSPF
  - OSPF System Tables
  - Example OSPF Installation
- Import - Export of Routing Information
- Advanced IP Features
  - IP Session Accounting
  - Network Address Translation
  - Proxy ARP
  - RIP Options

TCP/IP Primer

TCP (Transmission Control Protocol)/IP (Internet Protocol)[1] is often used to refer to the more general set of protocols known as the internet protocol suite. The protocols were designed to allow different types of computers and networks to communicate effectively. The internet protocol suite can be broken down into several layers, each of which provides/requires the services of an adjacent layer. These layers are often referred to as the TCP stack. The ordering and a brief description of each of these layers is shown below.



Note that each layer sends and receives information from adjacent layers. When a computer receives information from the network data passes upwards through the stack until it reaches the user's application. In the opposite direction; a user application sends data over the network, data moves downward through the stack until it reaches the physical network cabling.

Depending on where in the stack the data is and the direction it is moving, each layer performs applies additional information to or removes information from the packet. This mechanism is referred to as Encapsulation and is discussed in the next section.

1. This and following sections provides a greatly condensed discussion of TCP/IP. For detailed information the reader is referred to a comprehensive discussion of TCP/IP such as the *Internetworking wth TCP/IP* Series by Douglas E. Comer or *TCP/IP Illustrated* by W. Richard Stevens.

Encapsulation

The diagram on the following page shows the header information used at different layers when passing information between layers.

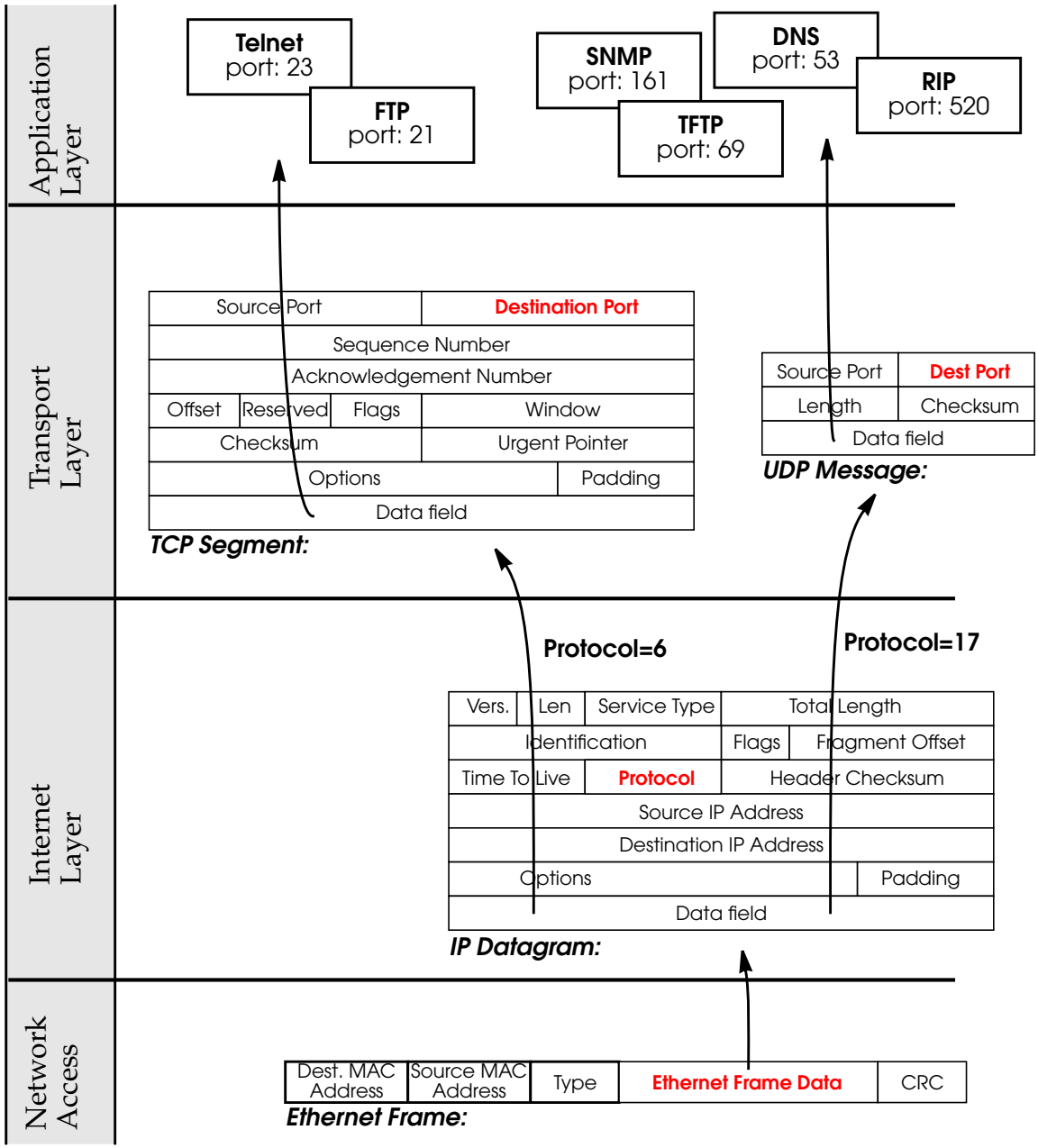
When information moves down the stack the sending layer applies control information to the data; this is referred to as header information. Each layer treats all information it receives from the layer above as data.

When information moves up the stack the receiving layer reads the header information (included by the sending computer), strips the header information away, and gives the leftover data to the next layer above. As infomratin moves up the stack each layer treats the information as header information and data combined.

- **Network Access Layer → Internet Layer**
  At this step, the contents of the ethernet frame's data field are simply passed to the Internet Layer. Note that the frame format used at this level may be slightly different (see section Ethernet Framing Types in Appendix B) but the concept is the same.
- **Internet Layer → Transport Layer**
  Here, the Internet Layer removes the IP header from the bytestream and decides which protocol in the Transport Layer to pass the data to using the value of the Protocol field.
- **Transport Layer → Aplpication Layer**
  The transport layer provides two types of very different services. The transport layer is responsible for passing the information to the proper port at the receiving host. This is determined by the contents of the destination port field.

  The TCP protocol is connection-oriented and provides error-detection and error-correction. Applications in the higher level layers requiring such services establish network connections using the TCP protocol.

  The UDP protocol is connection-less and provides a datagram delivery service. UDP based applications are message oriented and don't require the extensive services provided by TCP.


Application Layer
Transport Layer
Internet Layer
Network Access
Telnet port: 23
FTP port: 21
SNMP port: 161
TFTP port: 69
DNS port: 53
RIP port: 520
Source Port
Destination Port
Sequence Number
Acknowledgement Number
Offset
Reserved
Flags
Window
Checksum
Urgent Pointer
Options
Padding
Data field
TCP Segment:
Source Port
Dest Port
Length
Checksum
Data field
UDP Message:
Protocol=6
Protocol=17
Vers.
Len
Service Type
Total Length
Identification
Flags
Fragment Offset
Time To Live
Protocol
Header Checksum
Source IP Address
Destination IP Address
Options
Padding
Data field
IP Datagram:
Dest. MAC Address
Source MAC Address
Type
Ethernet Frame Data
CRC
Ethernet Frame:

IP Addressing

The Internet Protocol delivers packets to hosts using the Source and Destination host's Address fields found in the IP header. IP addresses consist of 4 octets, 8 bits/each totalling 32 bits. Addresses are commonly written in decimal form with each octet separated by dots (hence the term dot notation).

A typical IP address is 192.168.16.8, or
1100 0000.1010 1010.0001 0000.0000 1000 in binary.

An IP address consists of a network portion that identifies the network number and a host portion that identifies the host's number on that network. The location of the dividing line that separates the network portion from the host portion is different based on the network's "Class". There are 3 network classes which can be identified as follows:

Class	Octect 1 begins with	Octet 1	Octet 2	Octet 3	Octet 4
Class A	0...	< 128	1 - 254	1 - 254	1 - 254
		Networks	Hosts		
Class B	10...	129 - 191	1 - 254	1 - 254	1 - 254
		Networks		Hosts	
CLass C	110...	192 - 223	1 - 254	1 - 254	1 - 254
		Networks			Hosts

There is a 4th network class (Class D, octect 1 > 223) that is used for multicast addresses. Multicast addresses are used to address groups of computers that share a common protocol (as opposed to a common network) at one time.

Subnetting

Subnetting involves dividing an IP network into separate networks. It's often used to overcome topological constraints (cable lengths) or for organizational reasons (delegation of network management tasks).

Recall that a 32 bit IP address consists of a network portion and a host portion. Local sites can extend the meaning of the network portion to include some bits from the host's portion. Essentially this moves the dividing line between the network bits and the host bits creating additional networks but reducing the number of hosts on them.

To create a subnet each network host must use a 32 bit (4 octets) network mask, or "netmask". The bit values in the mask determine where the dividing line between the net and host portions are.

1. If the bit in the mask is **ON** (=1), the respective bit in the IP address belongs to the **NETWORK** portion.
2. If the bit in the mask is **OFF** (=0), the respective bit in the IP address belongs to the **HOST** portion.

This is where the standard network masks come from.

Class C Address:	**192** . (1100 0000)	**168** . (1010 1010)	**16** . (0001 0000)	**66** (0100 0010)
	← **Network** →			← **Host** →
Class C Netmask:	(1111 1111) **255** .	(1111 1111) **255** .	(1111 1111) **255** .	(0000 0000) **0**

A subnet mask commonly used on Class C networks is 255.255.255.192. This mask could be used to divide the 19.168.16.0 network into 4 subnets because the first two high order bits of the last octet are set. These 2 bits limit us to 4 possible subnets. This would include networks: 0, (00000000), 64 (01000000), 128 (10000000), and 192 (11000000).

Example Address:	**192** . (1100 0000)	**168** . (1010 1010)	**16** . (0001 0000)	**66** (0100 0010)
	← **Network** →			**Host**
Example Netmask:	(1111 1111) **255** .	(1111 1111) **255** .	(1111 1111) **255** .	(1100 0000) **192**

Once this netmask is applied, six bits of octet 4 are left over to identify the host. Six bits limit us to 64 (or 2^6) hosts per subnet. The example above identifies host number 2 ($00000010_2 = 2_{10}$).

The netmask above extends the network part to include the first two bits of octect 4 to identify the subnetwork. As stated above, 2 bits limits us to 4 subnets. The example above identifies subnetwork 64 ($01000000_2 = 64_{10}$).

So, the example address 192.168.16.66 when used with netmask 255.255.255.192, becomes equivalent to host 2 on subnet 192.168.16.64.

Protocols, Ports and Sockets

Together port numbers and protocol numbers identify a specific application (often referred to as a network service) on a host conputer.

The **protocol number** (the *protocol* field of an IP datagram) is an 8 bit number that identifies the transport protocol (UDP or TCP) in the Transport Layer. The Internet Layer uses this field when passing data up the stack. Some of the most commonly used protocol numbers include:

Number	Protocol and Name	
0	IP	Internet Protocol
1	ICMP	Internet Control Message Protocol
3	GGP	Gateway Gateway Protocol
6	TCP	Transmission Control Protocol
8	EGP	Exterior Gateway Protocol
12	PUP	PARC Universal Packet Protocol
17	UDP	User Datagram Protocol
20	HMP	Host Monitoring Protocol
22	XNS-IDP	Xerox NS IDP
27	RDP	Reliable Datagram Protocol
29	OSPF	Open Shortest Path Routing First

The current list of Protocol Numbers are contained in RFC 1700. This information is also available via the WWW from IANA (Internet Assigned Numbers Authority) via: ftp://ftp.isi.edu/in-notes/iana/assignments/protocol-nummbers

A **port number** is a 16 bit number that identifies an application in the Application Layer. The Transport Layer uses this number (the *destination port* field of the UDP message or TCP segment) when passing data up the stack.

Both a Source and a Destination Port field is present in the IP Datagram.

For IP packets moving up the TCP stack:
Src Port = port number of the sending application on remote host.
Dest Port = port number of the receiving application on the local host.

For IP packet moving down the stack:
Src Port = port number of the sending application on the local host.
Dest Port = port number of the receiving application on the remote host

The 16 bit port number defines a limit of 65,536 (2^{16}) possible port numbers. These 65,536 ports are divided as follows.

0 → 1023	1024 → 4999	5000→ 32767	32768 → 65535
priviledged	unprivileged		
server	clients	server	client

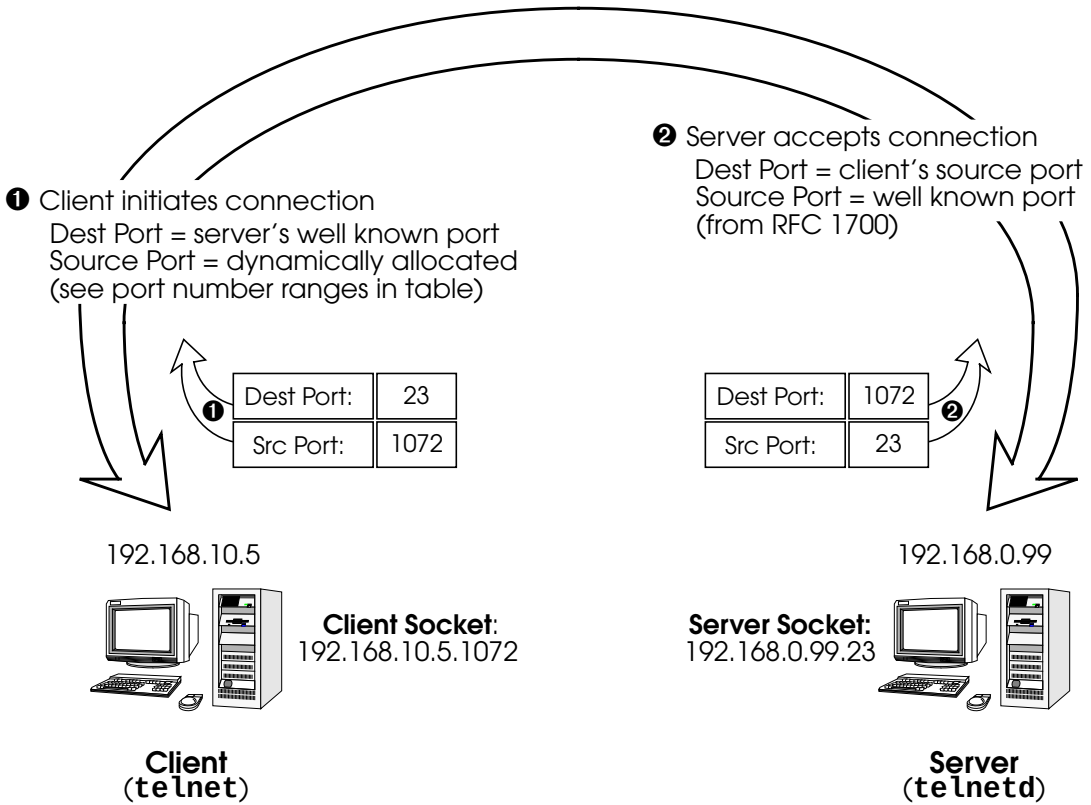
The *privileged* ports consist of standard port numbers, often referred to as "well known ports" that identify standard network services available on a computer;. The *unprivileged* ports are non-standard ports that may be defined by local hosts. Logically server port numbers are used by server applications and client ports by client applications. The assignment of port numbers will be made clear in the example network connection diagram that follows.

A few of the commonly used server port numbers are shown below.

Number	Port am Name	
21	FTP	File Transfer Protocol
23	telnet	The TELNET protocol/service
25	SMTP	Simple Mail Transfer Protocol
53	domain	Domain Name Service or DNS
80	HTTP	Hypertext Transmission Protocol
119	NNTP	Network News Transfer Protocol

The current list of Well Known Port Numbers are contained in RFC 1700. This information is also available via the WWW from IANA (Internet Assigned Numbers Authority) via: ftp;//ftp.isi.edu/in-notes/iana/assignments/port-numbers

A ***Socket*** identifies a specific network service on a computer (or other device). A socket consists of **IP Address**.**Port Number**. A computer at IP address 192.168.10.5 might provide a TELNET service at TCP port 23; the TCP socket is said to be: 192.168.10.5.23. Since many network services are multi-user applications a socket pair is required to identify a specific network connection. This socket pair consists of **Client Socket**:**Server Socket**. The diagram shown below shows how ports and sockets are used in a typical network connection.



IP Routing Protocols

In general, routing can be described as a method to determine the best interface to use when forwarding an incoming packet. The term "**best interface**" means selecting the interface from the router's routing table(s) that has the lowest cost. Cost is often measured by the number of intermediate stations the packet would pass through before reaching its destination.

The contents of the routing table may be configured statically. A router may optionally update its routing tables dynamically by exchanging information between other routers. This exchange of routing information is defined by a routing protocol.

Although all systems route data (PCs, workstations, routers) not all systems run a routing protocol. Some networks don't necessitate routing protocols —sites where routing information doesn't change or where only one route (or a set number of routes) exists.

Routing protocols allow a router to dynamically adapt to changing network conditions and to quickly make the best routing decision in complex networks. The two most commonly used (interior)[1] routing protocols; **RIP** and **OSPF** are covered briefly below.

RIP

With RIP (Routing Information Protocol) a router transmits and receives routing information among other routers. Approximately every 30 seconds a router broadcasts messages to adjacent networks using information from it's current routing table. This information consists of pairs of ***IP Address:Distance*** relationships. RIP determines a route's cost by the number of "hops" (distance) it takes for a packet to reach it's final destination. For this reason RIP is sometimes referred to as a distance vector algorithm.

By listening for information sent by other routers new routes and shorter paths for existing routes, are saved to the routing table when discovered via RIP. Because intermediate routes between networks may become unreachable, RIP also removes routes older than 5 minutes (i.e. routes that haven't been verified in the last 300 seconds).

OSPF

OSPF (Open Shortest Path First), is an interior routing protocol that is often used by larger network installations as an alternative to RIP. It was originally designed to ad-

1. The distinction between *Interior* and *Exterior* protocols is beyond the scope of this overview.

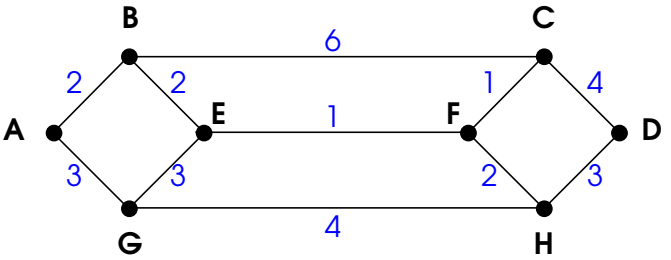
dress some of the limitations of RIP (when used in larger networks). Some of the problems (with RIP) that OSPF addresses include:

- **Faster Network Convergence**
  Changes in routing information are propagated immediately when changes occur and not periodically as with RIP.
- **Reduced Network Load**
  After a brief initialisation phase, routing information does not need to be refreshed as in RIP where the entire routing table is broadcast every 30 seconds.
- **Routing Authentication**
  Routers advertising OSPF routes can be authenticated.
- **Routing Traffic Control**
  OSPF areas can be closed to limit the amount of traffic resulting from routing advertisements.
- **Link-Costs**
  When calculating a route's cost OSPF can account for the different transport mediums such as LAN or WAN links.
- **No hop-count limitations**
  In RIP, routes spanning more than 15 hops are unreachable.

Although the OSPF protocol is more complex than RIP the basic concept is the same; the best interface must be calculated for forwarding packets to a particular station.

Shortest Path Routing

With RIP, routes are measured and selected according to number of hops it takes for a packet reach it's destination. In the diagram below, each node represents an IP router. According to RIP, the best route for a packet travelling from A to C will always be ABC.



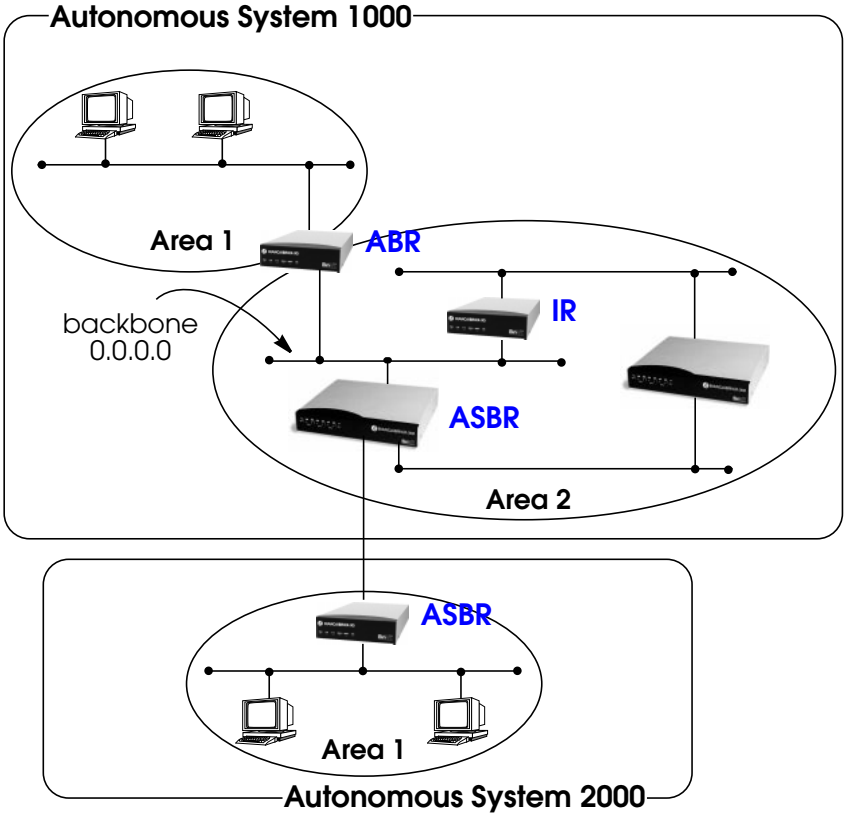
In OSPF each link has a cost associated with it (typically some fixed number divided by the bandwidth of the link). Routes are calculated and selected according to the least cost of the overall path a packet will travel. Thus in shortest-path routing the best path is also the fastest path (theoretically), regardless of the number of stations a packet travels through.

Assuming the relative costs of the links in the diagram above (shown in blue), according to OSPF the best route for a packet travelling from A to C is ABEFC (cost = 6). This route requires 4 hops as opposed to the 2 hop route (ABC) selected.

OSPF Routers and Link State Advertisement

OSPF is based on a concept of Areas. An Autonomous System (AS) consists of one or more Areas defined by network management. An Area may contain of one or more IP networks.

If an AS does contain more than one area one must be designated as the backbone, area: 0.0.0.0. All Area Border Routers (see Router Types) in an AS must have a physical connection to the backbone.

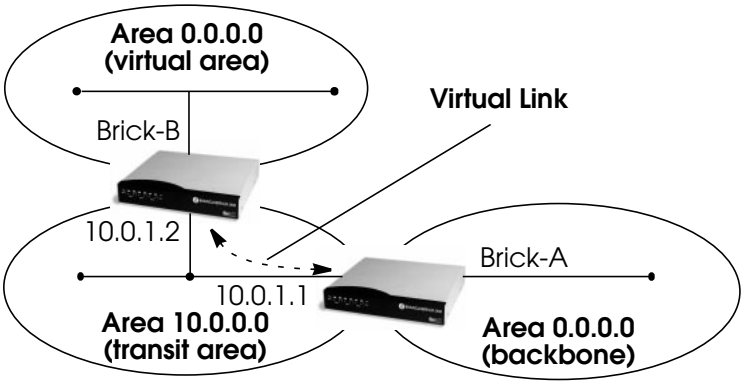


Any of the routers shown above could additionally be the Designated Router or Backup Designated Router for its respective network.

OSPF Virtual Links

Note that in OSPF the backbone, Area 0.0.0.0, is the center for all areas in the Autonomous System. However, sometimes it's not possible to physically connect all areas to the backbone. By configuring a "Virtual Link" between two area border routers a remote area an still be assigned to the backbone.

As shown in the diagram below. a virtual link is established between two Area Border Routers that share a common area; called the "transit area". Both routers must be physically connected to the backbone.



Router Types

The location of a router's interfaces with respect to an area determines the type of router it is and the types of Link State Advertisements it exchanges with other routers in that area.

- **Internal Routers** (IR) – A router whose interfaces are within the same area. All Internal Routers compute the shortest path tree to all destinations within its area.
- **Area Border Router** (ABR) – A router with interfaces in different areas but within the same autonomous system. Topological information is gathered (and stored) for each attached area allowing the ABR to compute the shortest path tree for each area separately.

- **Autonomous System Border Router** (ASBR) – A router that acts as a gateway between OSPF and external routes (i.e., routes provided by other routing protocols, static indirect routes, etc.). These routers propagate routes to external networks.
- **Designated Router** (DR) – On broadcast networks (token ring and ethernet) where more than two routers are present only the DR needs to synchronise its link state database with other routers.
- **Backup Designated Router** (BDR) – A backup router assumes the responsibilities performed by the DR if that system goes down.

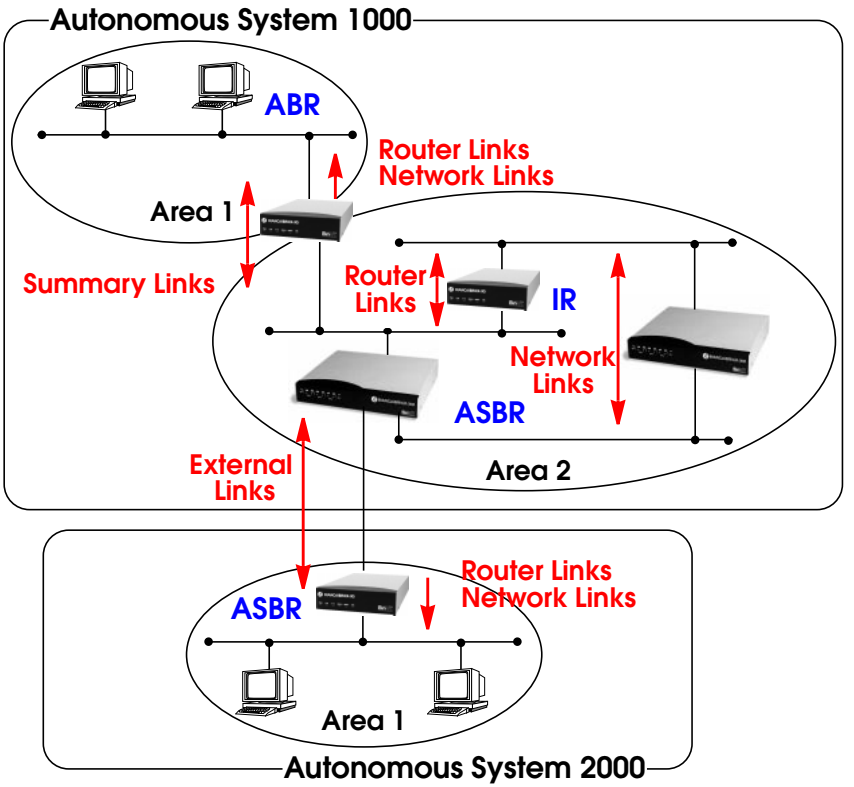
Link State Advertisement Types

OSPF routers exchange routing information via **Link-State Advertisements** (LSAs) that contain information about the networks that can be reached over the router's interfaces.

Link State Advertisements are broken down into five different types shown in the table below. The example network shown on the previous page is redisplayed below and shows where the different types of LSAs would be found in an OSPF network.

LSA Type	Purpose:
Router Links	**Generated by**: ALL OSPF Routers **Purpose**: Contains information regarding the state of a router's interfaces within a particular area. Router Links are only flooded within a single area.
Network Links	**Generated by**: The DR (or BDR). **Purpose**: Identifies all OSPF routers present on the network segment and their state. These links are only flooded within a single area.
Summary Links	**Generated by**: Area Border Routers **Purpose**: Identifies the presence of networks within an AS but outside the (local) area. Provides Inter-Area routes allowing routers to learn of networks in other Areas but within the AS.

LSA Type	Purpose:
ASBR Summary Links	**Generated by**: An Area Border Router. **Purpose**: A special type of summary link that provides routes to Autonomous System Border Routers allowing other routers in the AS to find their way out of the system.
External Links	**Generated by**: An Autonomous System Border Router. **Purpose**: Contains information about other Autonomous Systems and allows routers to learn about routes to networks there. External links are flooded into all areas except stub areas.



Router Identification

All OSPF routers in an Autonomous System must have a unique Router ID that identifies the router with respect to the AS. Generally an OSPF router's Router ID is taken to be the highest IP address for its first LAN interface.

Initialization

OSPF networks are said to be much "quieter" in comparison to RIP based networks. This is because in OSPF once the initialization phase is complete routing information is only exchanged when link state changes occur. This is much different than with RIP where every 30 seconds a router's complete routing table is broadcast and verified over the network.

The initialization phase of OSPF is completed once the Link State Database for the area has stabilized and generally occurs once:

1. The OSPF Neighbors have been identified.
2. The Designated and Backup Designated Routers have been established.

Neighbor Identification

When first coming into service an OSPF router attempts to identify its neighbor OSPF routers using the HELLO protocol. Two router are neighbors if they:

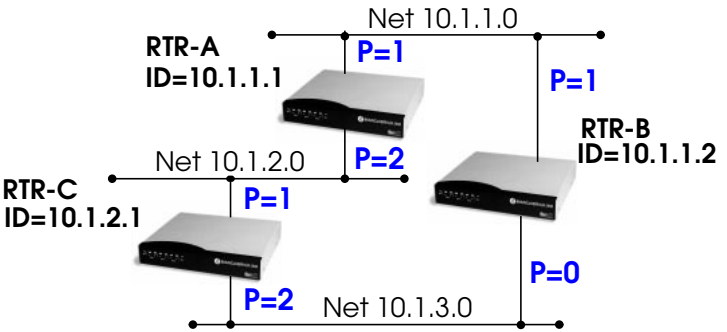
1. Share a common network.
2. Are using the same Area Number for that segment.
3. Are using the same Authentication for the segment.
4. Are using the same parameters (HELLO interval, etc.).

Neighbor routers then decide whether to synchronise their Link State Database (LSDB) with one another. All routers on the segment synchronise their LSDBs with the Designated Router (DR) and the Backup Designated Router (BDR).

Designated/Backup Designated Router Election

When Neighbor routers are identified (via the HELLO protocol) the DR and BDR are also identified. This is sometimes called DR and BDR election and is achieved via IP multicast packets which a router broadcasts via each network segment. For each seg-

ment the router with the highest OSPF priority generally becomes the DR. In case of a tie, the router with the higher Router ID becomes the DR.



The DR and BDRs for the three networks shown above would be elected as follows.

Network	DR	BDR
10.1.1.0	RTR-B	RTR-A
10.1.2.0	RTR-A	RTR-C
10.1.3.0	RTR-C	RTR-B

Building up the LSD and the STP

Link-State Advertisements, contain information about a routers interfaces (i.e.; link's IP address, mask, network type, networks reachable over the link, etc.).

All routers within an area receive all link-state information for all routers in the area. Once synchronized each router has an identical image of the link state database that describes the topological structure of the area.

This database allows each router to separately calculate a **shortest path tree** (SPT), using itself as the root, to any destination in the area. The SPT is used to determine the best interface to route packet. As in RIP the lowest cost route is used however the cost to a destination is calculated differently. In OSPF the cost (or metric) of a link is a function of the bandwidth provided by the link. The higher the bandwidth, the lower the cost.

Authentication

OSPF allows packets containing OSPF routing information to be individually authenticated. Two authentication methods are available which must be configured separately for each network segment.

1. Simple (password) authentication
   A simple text string is sent with each packet. This method is less secure since packet contents can be "sniffed" off the wire using a link analyzer.
2. MD5 (cryptographic) authentication
   When MD5 (Message Digest) is used each packet is appended with a 16 byte encrypted digest. The digest is a function of an authentication key and the contents of the packet. This method is more secure since the key is not sent with the packet.

Note: With MD5 authentication only the digest is encrypted and not the actual contents of the OSPF packet.

OSPF over Demand Circuits

Although OSPF generates less network traffic than RIP, the occasional exchange of routing information (HELLO packets, Link State Database updates or changes, etc.) can lead to increased costs for dial-up interfaces.

To help minimize these costs OSPF on the BRICK has been implemented to include special extensions for Demand Circuits as defined in RFC 1793, *OSPF over Demand Circuits*. These extensions allow for efficient use of dial-up interfaces with OSPF and avoiding excessive ISDN costs. In particular, this means:

1. The exchange of HELLO packets between neighbours is suppressed once the BRICK has synchronized its LSDB with that neighbour (A dial-up connection is initially opened to synchronize the database.).
2. Link State advertisements are only flooded to neighbour routers when an actual change needs to be propagated.
   Each LSA is marked with a special DoNotAge flag (identifiable by the DC-bit of the LSA or OSPF packet).

Note: This feature should only be used if all routers in the AS support this feature (RFC 1793) since some routers don't acknowledge the DC-bit (or use it differently). This could result in unwanted ISDN connections or connections.

Note: If a router without RFC 1793 support is removed from the domain in which this feature has been used it is recommended that all OSPF routers be briefly deactivated and re-activated to ensure that all LSAs generated by the removed router are actually flushed.

The Point-to-Point Protocol

The PPP (Point-to-Point Protocol) was designed as a standard method of communicating over point-to-point links. PPP actually consists of several underlying protocols, each of which perform a portion of the services offered by PPP.

In addition to HDLC (High level data link control) framing, PPP uses LCP (Link Control Protocol). LCP is used to negotiate options pertaining to the data link. Some of the options which can be negotiated using LCP are:

1. **Maximum-Receive-Unit**: The MRU specifies the maximum size of data packets to be processed over this link. The default value is 1500 bytes.
2. **Authentication-Protocol**: This option is used to specify which authentication procedure (CHAP or PAP), if any should be used for this link.
3. **Quality-Control**: This option specifies whether or not the quality of the link should be monitored.
4. **Protocol-Field-Compression**: This option specifies which, if any, protocol fields should be compressed over the link. Using this option could allow a higher throughput rate to be achieved.

Establishing a PPP connection

Establishing a PPP connection is accomplished step by step, in three simple phases.

1. Before any user data can be sent, the communicating partners must agree on which communications parameters the connection will use. This is accomplished using LCP mentioned earlier. Step by step, each side of the connection negotiates with the other to establish the best possible communications parameters.
2. The second phase is where the optional process are actually performed. This is where the authentication procedure (CHAP or PAP) would be performed if specified in phase 1. Additional parameters agreed upon in phase 1 are also performed here as well; i.e. if the Quality-Control option was agreed upon, a mechanism would then be started between the communicating partners, which helps to ensure a stable and secure connection.
3. The last phase of connection establishment involves making the connection available to the various network protocols. The actual closing of links is performed by LCP. However, as each network connection (multiple network connections are possible) closes, LCP may keep the physical connection open. PPP

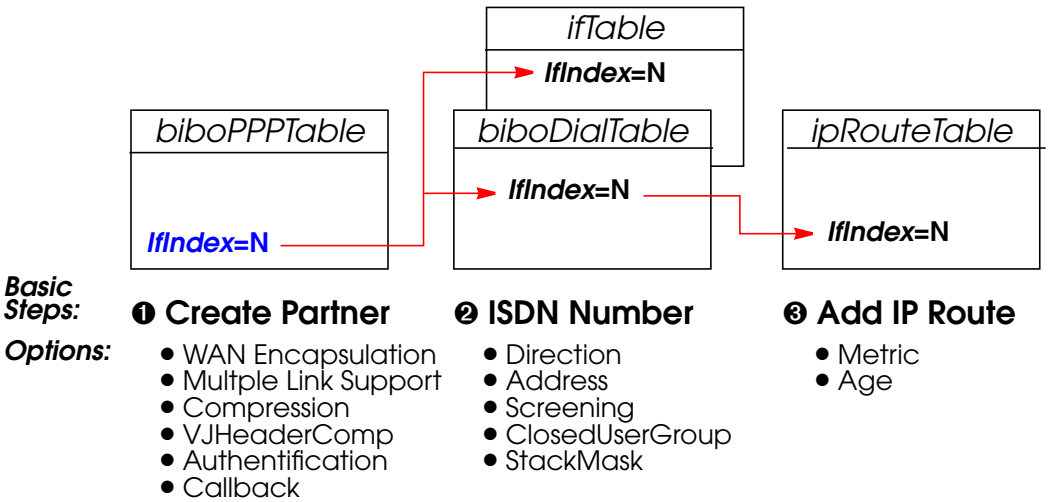
does not specify a default time limit to wait before automatically closing connections. Connections can be closed manually, or by setting a default wait time

DialUp IP Interfaces

Creating Dial-Up PPP interfaces on the BRICK basically involves three steps which correspond to creating the system tables shown below. Many different options are available when creating the respective table entries. An overview of the types of options available is shown in the diagram below.

- Create the PPP Partner Interface—***biboPPPTable***
- Identify the Partner's ISDN Number—***biboDialTable***
- Create an IP Route for the Partner—***ipRouteTable***

Note that after creating the partner interface in the ***biboPPPTable*** the BRICK generates a new ***ifIndex*** value and automatically creates an entry in the ***ifTable***. This ***ifIndex*** is very important since it identifies a specific software interface; it must be used when creating other system table entries to associate settings with the respective software interface.



An example SNMP shell session describing how to create a standard ISDN DialUp PPP interface is shown below. Most of the available optional settings mentioned above simply involve setting the respective variable to an appropriate non-default setting.

For a more detailed description of these optional settings and how to properly configure them, please refer to the section DialUp Options.

Creating a DialUp IP Interface

Step 1

The first step is to create the ***biboPPPTable*** entry. The only index variable in this table is the ***Type*** field; it defines this partner as either an **`isdn_dialup`** or **`leased`** line partner (Although leased line interfaces do appear here, leased-line partner interfaces can not be created using the ***biboPPPTable***.).

To create the table entry we can set this field and adjust the other entries as needed (objects not explicitly set revert to their default values).

```
mybrick: admin> biboPPPType=isdn_dialup
05: biboPPPType.1.5( rw):      isdn_dialup

mybrick : biboPPPTable > biboPPPTable
inx IfIndex(ro)              Type(*rw)          Encapsulation(-rw)
    Keepalive(rw)            Timeout(rw)        Compression(rw)
    Authentication(rw)       AuthIdent(rw)      AuthSecret(rw)
    IpAddress(rw)            RetryTime(rw)      BlockTime(rw)
    MaxRetries(rw)           ShortHold(rw)      InitConn(rw)
    MaxConn(rw)              MinConn(rw)        Callback(rw)
    Layer1Protocol(rw)       LoginString(rw)    VJHeaderComp(rw)
    Layer2Mode(rw)           DynShortHold       LocalIdent

05  10006                    isdn_dialup        ppp
    off                      3000               none
    none
    static                   4                  300
    5                        20                 1
    1                        1                  disabled
    data_64k                                    disabled
    auto                     0
```

The new dialup interface created above displays standard (default) setting consisting of the following characteristics:

Encapsulation	**`ppp`**	(See: WAN Encapsulation)
IP Address	**`static`** IP Address	(See: IP Address Settings)
Compression	**`none`**	(See: Compression)
Authentication	**`none`**	(See: Authentication)
MultiLinkSupport	**`1`** B-channel	(See: Mulitple Link Support)
ShortHold	**`20`** seconds	(See: Mulitple Link Support)

Callback	**disabled**	(See: ISDN Callback)
Layer1Protocol	**data_64k**	(See: Layer 1 Protocol)
LoginString	"empty"	(See: Auto-Login)
VJHeaderComp	**disabled**	(See: Header Compression)
Layer2Mode	**auto**	(See: Layer2Mode)
LocalIdent	"empty"	(See: PPP Identification)

Step 2

Next we need to define the partner's ISDN telephone number in the ***biboDialTable*** by associating it with the ***IfIndex*** created in step 1. As shown in step 1 display the contents of the ***biboPPPTable*** and locate the new interface index. The **inx** number for the new table entry is displayed to the screen when the entry is created. In most cases this will be the ***IfIndex*** field of the last table entry.

You may optionally verify this value is also present in the ***ifTable*** (in the ***Index*** field of the last table entry). In the example below the ISDN number 555 is associated with our new software interface 10006

```
mybrick: biboDialTable > biboDialIfIndex=10006 biboDialNumber=555

06: biboDialIfIndex.10006.6( rw):       10006
06: biboDialNumber.10006.6( rw):        "555"

mybrick: biboDialTable > biboDialTable

inx IfIndex(*rw)            Type(-rw)              Direction(rw)
    Number(rw)              Subaddress(rw)         ClosedUserGroup(rw)
    StkMask(rw)             Screening(rw)

06  10006                   isdn                   both
    "555"
    0xffffffff              dont_care

mybrick : biboDialTable >
```

Several options are also available in the Dial Table. Unless otherwise set, the following default values are used.

Type **isdn**
A Type of isdn_spv is used for a semi-permanent link and

is used in connection with the German 1TR6 protocol.

Direction **both**
Direction may be limited to **incoming** or **outgoing**.

ClosedUserGoups
Not used by default but may be set for sites receiving ISDN Closed User Groups services.

StkMask **0xffffffff** means all available ISDN stacks.

Screening **dont_care** (See: ISDN Screening)

Step 3

Now we need to create the appropriate routing table entry for this partner. One, possibly two, routing entries must be created in this step depending on whether a transfer network is being used . The example below assumes no transfer network is being used.

Using our partner's IP address (192.168.5.5) we add an indirect route to the partner's network. As before we need to associate this entry with the ***IfIndex*** for our partner interface from step 1 (10006).

```
mybrick: biboPPPTable > ipRouteIfIndex=10006 ipRouteDest=192.168.5.0
                                ipRouteType=indirect

03: ipRouteIfIndex.192.168.5.0.3( rw):  10006
03: ipRouteDest.192.168.5.0.3( rw):     192.168.5.0
03: ipRouteType.192.168.5.0.3(-rw):     indirect

mybrick: ipRouteTable> ipRouteTable

inx Dest(*rw)       IfIndex(rw)     Metric1(rw)     Metric2(rw)
    Metric3(rw)     Metric4(rw)     NextHop(rw)     Type(-rw)
    Proto(ro)       Age(rw)         Mask(rw)        Metric5(rw)
    Info(ro)

 03 192.168.5.0     10006           0               -1
    -1              -1              0.0.0.0         indirect
    netmgmt         355             255.255.255.0   -1
    .0.0

mybrick : ipRouteTable >
```

This route can also be created using the ifconfig command. (See: The ifconfig Command for command syntax).

DialUp Options

This section describes the various options found in the ***biboPPPTable***.

WAN Encapsulation

The ***biboPPPEncapsulation*** object defines the method used to encapsulation data packets transmitted over the ISDN link. The ISDN partner must also support the specified method for connections to be established. The type of encapsulation selected here also limits the types of protocols that can be routed over the interface. Possible encapsulation types and the protocols they support are shown below.

By default **ppp** encapsulation is used. Special information regarding some of the encapsulations (checkmarked in red) is contained below.

Note: In this table 5 means that the encapsulation may be configured but is not useful in most cases.

biboPPPEncapsulation	Supported Protocols			
	IP	IPX	Bridge	X.25
ppp	✓	✓	✓	
x25				✓
x25_ppp	✓	✓	✓	✓
ip_lapb	✓			
ip_hdlc	✓			
mpr_lapb	✓	✓	✓	
mpr_hdlc	✓	✓	✓	
frame_relay	✓	✓	✓	✓
x31_bchan				✓
x75_ppp	✓	×	×	
x75btx_ppp	✓	×	×	
x25_nosig				✓

biboPPPEncapsulation	Supported Protocols			
	IP	IPX	Bridge	X.25
x25_ppp_opt	✓	✓	✓	✓

Encapsulation: x75_ppp

x75_ppp encapsulation is used for asynchronous PPP over X.75 and is mainly used for accessing commercial service providers such as CompuServe Online Services. The *biboPPPLoginString* object is intended to be used with this encapsulation to automate the logon process with such service providers.

A typical logon string that might be used for logging onto Compuserve directly is shown below:

```
"-d1 \n e: CIS\n ID: 12345,6789/go:pppconnect\n
    word -d1 secret\n PPP"
```

Encapsulation: x75btx_ppp

x75btx_ppp encapsulation can be used to access CompuServe Online Services indirectly via the German Telekom's T-Online gateway. The *biboPPPLoginString* can be set to include the appropriate login information to automate the login process to the service provider.

A typical logon string that might be used for logging onto Compuserve via the T-Online gateway is shown below:

```
".n\ :000000 000327278259\n gabeseite 11 # # Name: CIS\n
    ID:12345,6789/go:pppconnect\n wor -d1 secret\n PPP"
```

Encapsulation: x25_nosig

x25_nosig (no signalling) encapsulation uses the same encapsulation method as x25. The only difference between the two is that with x25_nosig outgoing ISDN calls are not signalled as X.25 calls but as a data transfer call (DSS :Bearer Service unrestricted digital info without LLC).

Encapsulation: x25_ppp_opt

x25_ppp_opt encapsulation provides a special case of the *x25_ppp* encapsulation. It allows the BRICK to determine whether an incoming call is an X.25 call or a PPP call even if no outband authentication (by CLID) is possible. This is done by scanning the first incoming data packet.

Dial-in partners that can't be authenticated outband (CLID) are then given an X.25 connection via ISDN, or optionally a PPP connection, if they can be authenticated inband by using CHAP or PAP.

Once the dial-up connection is established only one protocool,, X.25 or IP, may be routed over the interface.

Note:

You will need one WAN partner definition for X.25, where the x25_ppp_opt encapsulation is selected, and one or more for PPP con nections (authentication via PAP, CHAP or RADIUS)

IP Address Settings

The ***biboPPPIpAddress*** object defines the BRICK's relationship to this host regarding its IP address. By default, **`static`** is used here. This assumes the PPP partner already has a fixed IP address configured and the appropriate IP routes (using this address) are already configured in the ***ipRouteTable***.

This object can also be set to dynamic_server or dynamic_client which is explained below.

dynamic_server This means the BRICK will attempt to assign this partner a new IP address at connection time. The next available IP address is retrieved from the ***biboPPPIpAssignTable***. If the dialup partner is configured to request a primary and/or secondary nameserver address, the BRICK responds by sending the current values of the ***biboAdmNameServer*** and ***biboAdmNameServ2*** objects.

dynamic_client This means the BRICK will accept its own IP address (for this dialup interface) from this partner at connection time. If not already set the BRICK requests the primary and/or secondary nameservers address. If the dialup partner provides this information the BRICK sets the ***biboAdmNameServer*** and/or ***biboAdmNameServ2*** objects.

Compression

The ***biboPPPCompression*** object defines the type of data compression (performed in software) to use with this partner. The BRICK-XS, BRICK-XM, and V!CAS support both STAC and V42bis data compression. On the BRICK-XL V42bis data compression is supported in software; STAC compression will be performed in hardware via an additional feature module available in a future release.

Data compression can only be used in connection with the ***Encapsulation*** settings shown below. Although its possible to configure any Compression–Encapsulation combination in the ***biboPPPTable,*** compression over the link will only be achieved when configured as follows.

biboPPPCompression	*biboPPPEncapsulation*
stac	ppp
stac	x25_ppp
v42bis	mpr_lapb
v42bis	ip_lapb

STAC compression is supported according to RFC 1974 and 1962 (PPP *Stac LZS Compression* and *PPP Compression Conrol Protocol* respectively) standards, which, depending on the data can increase performance variably. Typically, performance is increased by a factor of 2 to 3; with the best case scenario at a factor of 30. The Stacker LZS algorithm is developed by Hi/fn Inc.

STAC compresssion on the BRICK is also compatible with Cisco's proprietary STAC implementation which is automatically detected at connection time.

Note:

Due to heavy system requirements made by this algorithm only 4 instances of can be used simultaneously, for example,
4 partner connections @ 1 B-Channel each, OR
2 partner connections @ 2 B-Channels each, etc.
This limit does not affect the BRICK-XS or V!CAS products.

Authentication

The ***biboPPPAuthentication*** determines the type of authentification to use when establishing dialup connections with this partner. Three types of authentification meth-

ods are available here; **chap**, **pap**, and **radius**. The value **both** can also be set and means that both PAP and CHAP should be used.

Mulitple Link Support

Multiple Link Support allows data connections to and from dialup ISDN partners to be run over multiple channels concurrently. By dynamically allocating bandwidth (automatically openning and closing additional channels) greater throughtput rates can be acheived when needed. For dialup ISDN connections this of course can lead to increased costs.

Every 5 seconds the BRICK calculates the current throughput for each dialup interface that is open. When throughput rises above a preset upper bound additional ISDN channels are opened. If throughput drops below a specified level unneeded channels are closed.

Using the following fields of the ***biboPPPTable*** the BRICK determines how multiple link support should be handled for the specified partner.

InitConn InitConn defines the number of ISDN channels to initially open when a connection is established with this partner. By default 1 B-Channel is opened.

MaxConn MaxConn defines the maximum number of channels to have open at any given time for connections to this partner. By default the max number of chanels is 1.

MinConn MinConn defines the minimum number of channels to keep open with this partner. If throughput drops, the number of open channels will never become less than this value. The only exception is when ***ShortHold*** (or ***DynShortHold***; see Short Hold) timer runs out. By default 1 channel is always kept open.

Short Hold

Short Hold means that an existing ISDN connection can be automatically taken down by waiting a specified (configurable) amount of time once the line becomes silent. Silent here means that for the adjusted time no more data packets have been going out. Data, which is generated by the BRICK itsself cyclicly, like for example RIP broadcasts and KeepAlives are not considered.The BRICK supports two types of Short Hold, Stat-

ic and Dynamic. Note that Dynamic Short Hold can only be used if the ISDN AOCD[1] (advice of charge during the call) feature is activated.

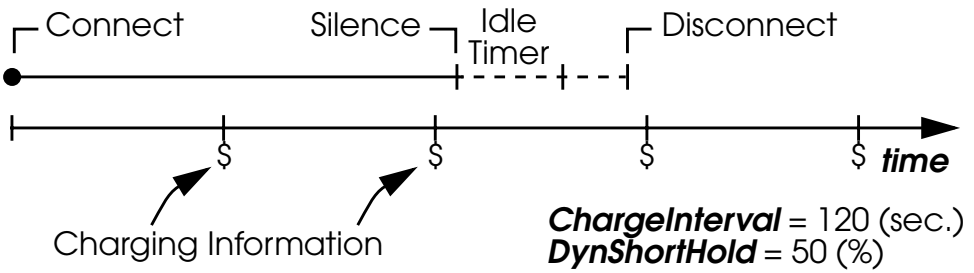
Static Short Hold

Static Short Hold and involves setting the ***biboPPPShortHold*** variable to the amount of time (in seconds) to wait before disconnecting the line. Though less flexible than Dynamic method static short hod can always be used.

Dynamic Short Hold

Dynamic short hold provides greater flexibility in determining when the line is taken down. Here the ***biboPPPDynShortHold*** variable is used. This object defines the percentage of the current Charging Interval (sent by the ISDN and saved on the BRICK in the ***biboPPPChargeInterval*** object) to wait before closing the link.

For example, if ***biboPPPDynShortHold*** is set to 50 (%), and the last measured ***biboPPPChargeInterval*** was 120 seconds, the idle timer is set to 60 seconds. If the ChargeInterval length changes (weekday/weekend, time of day, etc.) the idle timer setting adjusts accordingly.



Recommended Dynamic Short Hold Settings:

- For ***interactive connections*** (e.g. telnet) you should specify a rather high Dynamic Short Hold percentage (e.g. 80-90) to avoid frequent disconnects due to short periods of inactivity.

1. Called »*Übermittlung der Tarifeinheiten* während *der Verbindung*« in Germany

- For ***internet connections*** (WWW, http, etc.) you should specify a medium to high Dynamic Short Hold percentage (e.g. 50-80) to avoid frequent disconnects due to waiting periods.
- For ***data connections*** (e.g. ftp) you should specify a low Dynamic Short Hold percentage (e.g. 10-40) to avoid unnecessarily waiting—and incurring charges—once a transfer is complete.

Note:

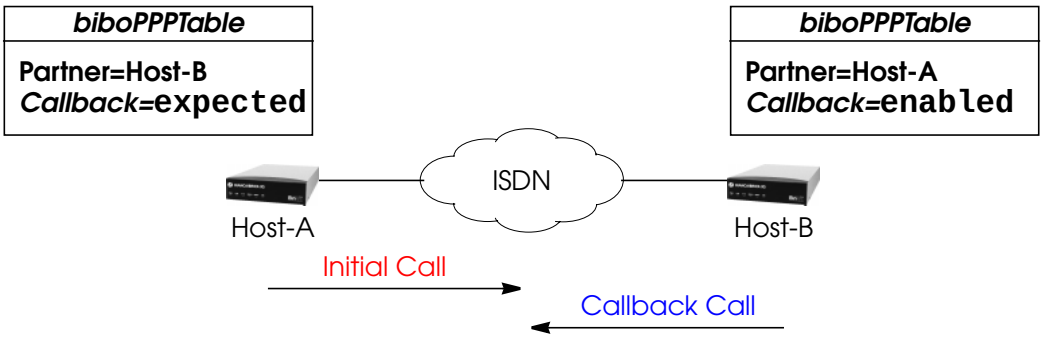
If configured, the Static Short Hold timer will *always* take precedence over Dynamic Short Hold to avoid permanent connections.
Make sure to set the Static Short Hold to a value greater than the length of a charging unit if you want Dynamic Short Hold to have any effect.
For example, in Germany there are different maximum charging unit lengths for different tariff zones (City = 4 minutes, long distance calls = 2 minutes), so you can set the *Static* Short Hold to 245 (>4 minutes) for City connections, and to 125 (>2 minutes) for long distance calls, to avoid nullifying your Dynamic Short Hold settings.

Note:

If you are using Dynamic Short Hold in connection with channel bundling, please note that the channels are released one by one, keeping open each channel until shortly before the next advice of charge is expected for this channel, thus maximizing the connection time without further cost. The call will of course be disconnected immediately if either side actively closes it.

ISDN Callback

ISDN callback operation is supported in both directions on the BRICK. Using the ***biboPPPCallback*** object ISDN callback can be configured separately for each PPP partner in either **`enabled`** or **`expected`** mode.



Callback: `expected`

Expected mode operates as follows:

1. The BRICK places an initial call to the ISDN partner. The partner may be another BRICK (configured for enabled mode) or an other system that supports ISDN callback.
2. The remote partner closes the initial connection and returns the ISDN call; hence callback is "expected" from this partner.

For the Initial Call at least one ISDN number (not containing wildcards) must be present in the ***biboDialTable*** (***Direction*** is either **`outgoing`** or **`both`**). The first number found for this partner (Host-B above) is used to place the call.

For the Callback Call to be accepted by the receiving host an incoming number entry must be present for calling partner in the ***biboDialTable***. This entry may not contain wildcard characters.

`Callback: enabled`

Enabled mode operates as follows:

1. The BRICK receives an ISDN call from this partner.

2. After authenticating the caller (via Calling Line ID or CHAP/PAP) the BRICK closes the initial connection and places a new call to the partner.

For the Initial Call to be acknowledged on the receiving host an incoming number entry (***Direction*** = **`incoming`** or **`both`**) must be present for the calling partner in the ***biboDialTable*** and may not contain wildcard characters.

To place the Callback Call an outgoing number (not containing wildcards) must be present for this partner in the ***biboDialTable*** (***Direction*** is either **`outgoing`** or **`both`**). If the Callback Call is not successful the BRICK waits a preset amount of time and re-attempts callback up to ***biboPPPMaxRetries*** times (by default 5 attempts are made).

Layer 1 Protocol

The ***biboPPPLayer1Protocol*** object defines the layer 1 protocol to use for connections to/from this dialup partner. By default data_64k is selected. The list of possible layer 1 settings is shown below. .

Layer1Protocol	Comment
`data_64k`	Default setting.
`data_56k`	For connections over ISDN lines limited to 56k lbandwidth (e.g., calls to/from North America).
`modem`	The actual layer 1 connection parameters are negotiated by the calling/receiving modems[a].
`modem_profile_1` – **`modem_profile_8`**	The incoming/outgoing call to the specified partner uses the modem settings defined in the respectve profile. Refer to the ***mdmProfileTable***.[a]
`dovb`	Special setting for "Data over Voice Bearer"[b]
`v110_1200` – **`v110_38400`**	V.110 bit rate adaptation. Identifies the settings to use (1200 baud, 8, N, 1 through 38400 baud, 8, N, 1) for calls to this partner.

a.Only for products with internal modems (BRICK-XL, V!CAS and XS-Office).
b.Used mainly in N.America to allow data transfers over voice circuits (i.e., the digital call is initially setup using voice signalling).

Auto-Login

The ***biboPPPLoginString*** defines a text string that is used to automatically log into the called system. This variable is only useful in connection with the x75_btx and x75_ppp encapsulations (see Encapsulation: x75_ppp) for automating dialup connections with CompuServe Online Services.

The Login String consists of special characters and alternating expect – send sequences separated by spaces. The first string detected as not being a secial character is assumed to be an expect string. Currently the following special characters are recognized.

Special TAG	Meaning
-d<*number*>	Indicates a pause of <*number*> seconds.
\n	Indicates transmit a carriage return.

A typical logon string that might be used for logging onto Compuserve directly is shown below:

```
“-d1 \n e: CIS\n ID: 12345,6789/go:pppconnect\n
    word -d1 secret\n PPP”
```

Once the initial connection is established this string would be used to:

Wait 1 second
    transmit a carriage return
expect the string: “e:”
    transmit “CIS” followed by a carriage return
expect the “ID:” string
    transmit “12345,6789/go:pppconnect” and a carriage return
expect the string: “word”
wait 1 second
    then transmit “secret” followed by a carriage return
expect the string “PPP”

The Compuserve UserID and Password shown above (12345,6789 and secret), would have to be changed of course.

Header Compression

The ***biboPPPVJHeaderComp*** object defines whether Van Jacobson TCP/IP header compression (VJHC) should be used with this partner. For IP capable interfaces VJ-HeaderComp may be set to either **`enabled`** or **`disabled`**.

If the dialup partner supports header compression this option can be used to help reduce the size of TCP/IP packets and provide improved performance (line efficiency). VJHC is supported acording to RFC 1144.

Compression settings are negotiated at connection time during PP setup. If the called party is does not support VJHC (or if disabled for this partner but the calling party requests it) the link is still established, but without header compression enabled. When negotiated sucessfully a system message is generated in the syslogTable (Subject=ppp, Level=info).

Layer2Mode

The ***biboPPPLayer2Mode*** object defines the mode to use at layer 2 for connections to this dialup partner. For leased line partners the layer 2 mode set in the ***Type*** field of the ***isdnChTable*** is used.

Layer2Mode is only relevant, if ***biboPPPEncapsulation*** involves a LAPB based protocol which is the case for the following settings:

`x25`	**`x25_ppp`**	**`x25_ppp_opt`**	**`ip_lapb`**
`mpr_lapb`	**`x31_bchan`**	**`x75_ppp`**	**`x75btx_ppp`**
`x25_nosig`			

By default ***Layer2Mode*** is set to **`auto`**. This means that the BRICK will adjust the its layer 2 mode appropriately depending on the direction of the call for this partner. For an incoming call from this partner the BRICK operates a DCE, when placing a call to this partner the BRICK operates as DTE.

Setting this object to either **`dte`** or **`dce`** means the BRICK will always operate as DTE or DCE respectively, regardless of the direction of the call. Also if dte or dce is set an approriate entry in the ***biboDialTable*** must also be present.

if *biboPPPLayer2Mode* =	if *biboDialDirection* =
`dte`	**`both or outgoing`**

if *biboPPPLayer2Mode* =	if *biboDialDirection* =
dce	both or incoming

PPP Identification

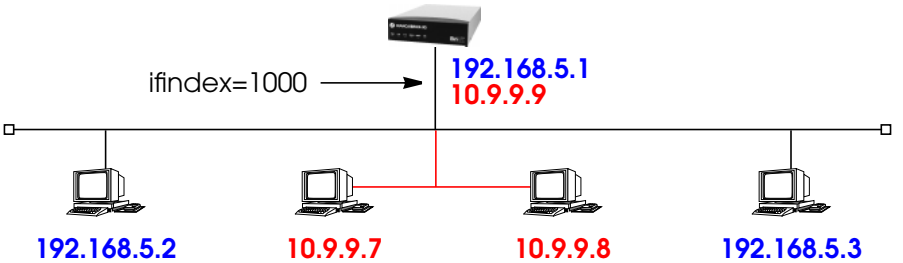
When PAP and/or CHAP authentication is used with the dial-up partner the BRICK must identify itself with a special string known as the PPP ID.

When authentication is performed with this partner the BRICK sends the value of the ***biboPPPLocalIdent*** varialbe. If this object is not set, the BRICK uses the contents of the ***biboAdmLocalPPPIdent*** variable in the ***admin*** tabe.

Dual IP Address Interfaces

Normally each (physical) BRICK ethernet interface is assigned a single IP address. This address can be seen in the BRICK's ***ipAddrTable*** which lists the current IP address for all BRICK interfaces.

A second IP address may be assigned to an ethernet interface by creating a direct route in the ***ipRouteTable*** that points to the interface.



The ethernet interface for the BRICK in the diagram (assumed to already be assigned 192.168.5.1) could be assigned a second address by adding the following IP route.

```
mybrick: system > ipRouteIfIndex=1000 ipRouteDest=10.9.9.0 ipRouteNextHop=10.9.9.9

01: ipRouteDest.10.9.9.0.2( rw): :        10.9.9.0
01: ipRouteNextHop.10.9.9.0.2( rw): :     10.9.9.9
01: ipRouteIfIndex.10.9.9.0.2( rw): :     1000

mybrick: ipRouteTable> ipRouteTable

inx Dest(*rw)          IfIndex(rw)       Metric1(rw)       Metric2(rw)
    Metric3(rw)        Metric4(rw)       NextHop(rw)       Type(-rw)
    Proto(ro)          Age(rw)           Mask(rw)          Metric5(rw)
    Info(ro)

01  10.0.0.0           1000              0                 -1
    -1                 -1                10.9.9.9          direct
    netmgmt            5                 255.0.0.0         -1
    .0.0

mybrick : ipRouteTable >
```

Both IP addresses will appear in the ***ipAddrTable***.

```
mybrick: ipRouteTable > ipAddrTable

inx Addr(*ro)          IfIndex(ro)       NetMask(ro)       BcastAddr(ro)
    ReasmMaxSize(ro)

00  192.168.5.1        1000              255.255.255.0     1
    65535
01  10.9.9.9           1000              255.0.0.0         1
    65535

mybrick : ipAddrTable >
```

The BRICK can now route between the two networks.

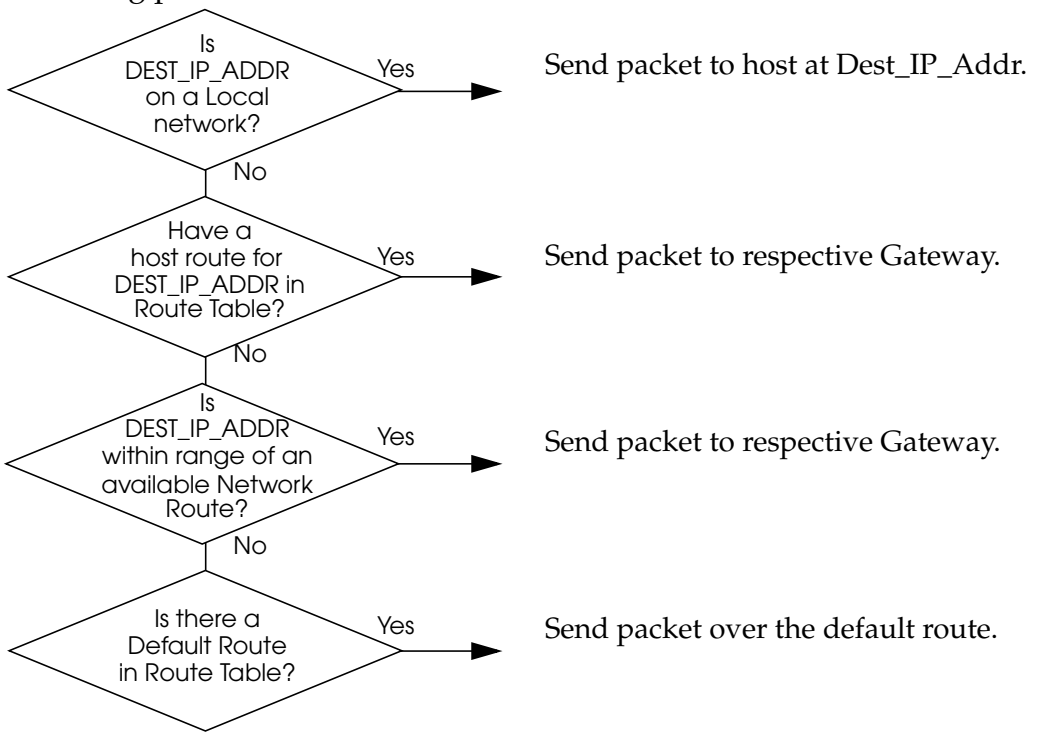
IP Routing on the BRICK

The BRICK's IP routing table is contained in the ***ipRouteTable***. It contains, among other information, a list of destination addresses (host or network) and gateway addresses to be used when routing IP packets to those destinations. When the routing table is kept current, the BRICK is prepared to make intelligent decisions as to where to route incoming packets.

Before any IP packets can be routed the BRICK first determine:

1. The Destination IP Address (DEST_IP_ADDR) from the IP packet.
2. The Destination Network (DEST_ADDR) from DEST_IP_ADDR.
3. Whether a default route exists.

The BRICK can then decide which interface to route the packet over. To make this decision the BRICK uses a rather complicated internal routing algorithm. The general routing algorithm might proceed as shown below. Routing involves doing this for each packet that arriveing packet.



Extended IP Routing

Most routing decisions are based solely on a packet's destination address, Extended IP routing allows you to route IP traffic based on additional information. Extended routes are configured in the ***ipExtRtTable***. Each extended route table entry defines a separate route which can be separately or jointly based on:

- Contents of the IP packet header.
- The source interface the packet arrived on.
- The current state of a BRICK interface (normally a dialup interface).

Table Field		Global	Meaning
Contents of IP Header.	***Protocol***	dont_verify	Protocol field of IP header
	SrcAddr	0.0.0.0	Source Address field of IP header.
	SrcMask	0.0.0.0	Used with SrcAddr.
	SrcPort	-1	Source Port field of IP header
	SrcPortRange	-1	If not = -1 last number of range of ports, starting from SrcPort.
	DstAddr	0.0.0.0	Destination Addr field of IP header
	DstAddrMask	0.0.0.0	Used together with DstAddr.
	DstPort	-1	Destination Port field of IP header.
	DstPortRange	-1	Used together with DstPort field.
	Tos	0	Type of Service field of IP header.
	TosMask	0	Type of Service field of IP header.
BRICK Interface	***DstIfMode***	-	The state of the DstIfIndex.
	SrcIfIndex	-0	The interface to route packet to.

Route Priority

When routing IP packets, the BRICK always checks for extended routes first. If the ***ipExtRtTable*** is empty, or a matching entry is not found, the ***ipRouteTable*** is consulted.

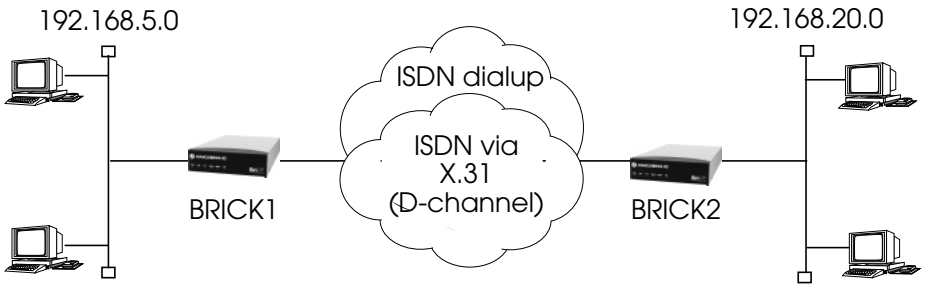
1. First check ***ipExtRtTable***; if a route is found, route packet, otherwise
2. Check ***ipRouteTable***.

NOTE:

If more that one route is found in a routing table, the route with the lowest metric value specified in the ***Metric1*** field is used. If multiple routes are found with the same metric, it is not possible to determine which route will be used.
The ***Metric2***, ***Metric3***, ***Metric4***, and ***Metric5*** fields are not used.

Configuring Extended Routes

For example the two LANs shown below could be connected via an ISDN basic rate interface. For telnet sessions we might want to take advantage of volume-based charging of X.31 (X.25 in the D-channel) and avoid the much higher costs for ISDN dialup connections. All other IP traffic could continue to use the dialup ISDN link.



Presetting

1. Each BRICK needs to be configured to allow for normal routing via our ISDN dialup link (dialup1, ifindex=10001).
2. Next, we'll need to create an MPX25 (mpx1, ifindex=20001) interface to allow IP traffic to be routed over X.25.

Configuration

Since IP is already routed between the LANs using our dialup interface, we only need to filter out the telnet traffic. This can be done using the *Protocol*, *SrcPort*, and *DstPort* variables in the ***ipExtRtTable***.

Step 1

For BRICK1 the extended IP routes would be added as follows. The first route is for IP packets destined for hosts on network 192.168.20.0 with a source IP port of 23. This is for outgoing telnet sessions.

The second route is for IP packets destined for hosts on the remote network, with a destination IP port of 23. This is for the incoming telnet sessions.

```
brick1: system > ipExtRtTable

inx Protocol(*rw)     SrcIfIndex(rw)      SrcAddr(rw)      SrcMask(rw)
    SrcPort(rw)       SrcPortRange(rw)    DstAddr(rw)      DstMask(rw)
    DstPort(rw)       DstPortRange(rw)    Tos(rw)          TosMask(rw)
    DstIfMode(rw)     DstIfIndex(rw)      NextHop(rw)      Type(-rw)
    Metric1(rw)       Metric2(rw)         Metric3(rw)      Metric4(rw)
    Metric5(rw)       Proto(rw)           Age(rw)

brick1: ipExtRtTable> Protocol=tcp SrcPort=23 DstAddr=192.168.20.0 DstIfIndex=20001

brick1: ipExtRtTable> Protocol=tcp DstPort=23 DstAddr=192.168.20.0 DstIfIndex=20001

inxProtocol(*rw)  SrcIfIndex(rw)      SrcAddr(rw)      SrcMask(rw)
    SrcPort(rw)       SrcPortRange(rw)    DstAddr(rw)      DstMask(rw)
    DstPort(rw)       DstPortRange(rw)    Tos(rw)          TosMask(rw)
    DstIfMode(rw)     DstIfIndex(rw)      NextHop(rw)      Type(-rw)
    Metric1(rw)       Metric2(rw)         Metric3(rw)      Metric4(rw)
    Metric5(rw)       Proto(rw)           Age(rw)

00  tcp               0                   0.0.0.0          0.0.0.0
    23                -1                  192.168.20.0     255.255.255.0
    -1                -1                  0                0
    dialup_wait       20001               0.0.0.0          indirect
    0                 0                   0                0
    0                 netmgmt             0 00:20:25.00

 01 tcp               0                   0.0.0.0          0.0.0.0
    -1                -1                  192.168.20.0     255.255.255.0
    23                -1                  0                0
    dialup_wait       20001               0.0.0.0          indirect
    0                 0                   0                0
    0                 netmgmt             0 00:20:26.00

brick1 : ipExtRtTable >
```

Step 2

The same extended routes would also be added to BRICK2. The ifIndex for our MPX25 partner (BRICK1) is assumed to be 20003 on BRICK2. Again the first route is for the outgoing telnet sessions, the second is for the incoming sessions.

```
brick2 : ipExtRtTable > Protocol=tcp SrcPort=23 DstAddr=192.168.5.0 DstIfIndex=20003

brick2 : ipExtRtTable > Protocol=tcp DstPort=23 DstAddr=192.168.5.0 DstIfIndex=20003

brick2 : ipExtRtTable > ipExtRtTable

inxProtocol(*rw)  SrcIfIndex(rw)       SrcAddr(rw)      SrcMask(rw)
    SrcPort(rw)       SrcPortRange(rw)   DstAddr(rw)      DstMask(rw)
    DstPort(rw)       DstPortRange(rw)   Tos(rw)          TosMask(rw)
    DstIfMode(rw)     DstIfIndex(rw)     NextHop(rw)      Type(-rw)
    Metric1(rw)       Metric2(rw)        Metric3(rw)      Metric4(rw)
    Metric5(rw)       Proto(rw)          Age(rw)

00 tcp                0                  0.0.0.0          0.0.0.0
    23                -1                 192.168.5.0      255.255.255.0
    -1                -1                 0                0
    dialup_wait       20003              0.0.0.0          indirect
    0                 0                  0                0
    0                 netmgmt            0 00:20:25.00

 01 tcp               0                  0.0.0.0          0.0.0.0
    -1                -1                 192.168.5.0      255.255.255.0
    23                -1                 0                0
    dialup_wait       20003              0.0.0.0          indirect
    0                 0                  0                0
    0                 netmgmt            0 00:20:26.00

brick2 : ipExtRtTable > ipExtRtTable
```

Additional Options

A range of ports can also be specified using the ***SrcPort*** and ***SrcPortRange*** variables together. ***SrcPort*** specifies the first port number in the range, ***SrcPortRange*** defines the last port in the range. Both ports are included in the range. For example, the extended routes in the above examples could have looked like this:

Protocol=tcp SrcPort=21 SrcPortRange=23
DstAddr=192.168.5.0 DstIfIndex=20003

Protocol=tcp DstPort=21 DstPortRange=23 DstAddr=192.168.5.0 DstIfIndex=20003

allowing ftp and telnet sessions (IP ports 21 and 23) to be routed over a specific interface.

BOOTP and DHCP

BootP and DHCP (Dynamic Host Configuration Protocol) are two methods that are commonly used by a host to retrieve important configuration information from a remote host on the LAN. For diskless workstations this might be it's IP address and netmask but could also include other information such as a nameserver's address to use. The BRICK supports both protocols.

BootP The BRICK can operate as a **BootP Relay Agent**. This means BootP requests received over the BRICK's interfaces are forwarded to the BootP Server (***biboAdmBootpRelayServer***) if one exists.
To configure a Relay Agent see BootP Relay Agent Settings.

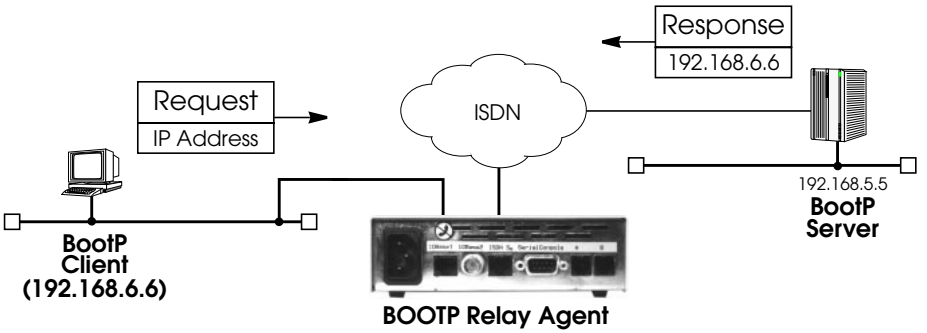
NOTE: The BRICK can retrieve it's own configuration information at boot time via a BootP server. See the section BOOT Options on the BRICK, in Chapter 4 System Administration on the BRICK.

DHCP The BRICK can also operate as a **DHCP Server.** DHCP is intended to make maintenance of remote and/or diskless workstations easier. It is also commonly used on Windows based systems. The major benefit of DHCP is that it gives the network manager the ability to manage a limited number of IP addresses among several hosts. The DHCP server on the BRICK provides the extra benefits of accessibility.
To configure a DHCP Server see DHCP Server Setting.

NOTE: BootP and DHCP use the same message format. When a DHCP message is received on the BRICK but the DHCP server is NOT configured the DHCP request is automatically forwarded to the BootP server if one is configured.
Although BootP and DHCP use the same message format, they are not completely compatible. Clients that support BootP can't be configured via the BRICK's DHCP server.

BootP Relay Agent Settings

The BRICK forwards all BootP requests received over it's LAN interfaces to the host defined in the ***biboAdmBootpRelayServer*** field of the ***admin*** table.



BootP Relaying for this example would be configured as follows:

Step 1

First, configure the IP address for the BootP Server in the admin table.

```
mybrick: > admin
...
biboAdmBootpRelayServer(rw):       0.0.0.0
...
mybrick: admin> biboAdmBootpRelayServer=192.168.5.5
biboAdmBootpRelayServer(rw):       192.168.5.5

mybrick: admin>
```

Step 2

Before the BRICK can begin forwarding requests to this host a partner interface must be created in the ***biboPPPTable.*** Below we create a standard dial-up interface for the server.

```
mybrick: admin> biboPPPType=isdn_dialup

05: biboPPPType.1.5(rw):      isdn_dialup

mybrick : biboPPPTable > biboPPPTable

inx IfIndex(ro)              Type(*rw)              Encapsulation(-rw)
    Keepalive(rw)            Timeout(rw)            Compression(rw)
    Authentication(rw)       AuthIdent(rw)          AuthSecret(rw)
    IpAddress(rw)            RetryTime(rw)          BlockTime(rw)
    MaxRetries(rw)           ShortHold(rw)          InitConn(rw)
    MaxConn(rw)              MinConn(rw)            Callback(rw)
    Layer1Protocol(rw)       LoginString(rw)

05  10006                    isdn_dialup            ppp
    off                      3000                   none
    none
    static                   4                      300
    5                        20                     1
    1                        1                      disabled
    data_64k

mybrick: biboPPPTable >
```

Step 3

Don't forget to add the telephone number to the BRICK's Dial table.

```
mybrick: biboPPPTable > biboDialIfIndex=10006 biboDialNumber=555

06: biboDialIfIndex.10006.6(rw):      10006
06: biboDialNumber.10006.6(rw):       "555"

mybrick : biboDialTable >
```

For information on other options when creating partner interfaces, refer to section Creating a DialUp IP Interface.

DHCP Server Setting

DHCP on the BRICK consists of the ***ipDhcpTable*** and ***ipDhcpInUseTable***.

ipDhcpTable Used to define a pool of addresses the BRICK will use when assigning IP addresses to DHCP clients. Each entry in the table defines a range of addresses to use for requests received on the respective interface.

ipDhcpInUseTable Displays which addresses are in use (by host's MAC address) as well the address' expire time.

In Setup Tool the Configuration of DHCP can be entered in the **DHCP** submenu of the **IP** menu.

A range of 13 addresses for the BRICK's first LAN interface could be configured as follows:

```
mybrick: ipDhcpTable> IfIndex=1000 First=192.168.5.80 Range=9
...
mybrick: ipDhcpTable> IfIndex=1000 First=192.168.5.90 Range=4
...
mybrick: ipDhcpTable> ipDhcpTable

inx IfIndex(*ro)        State(-rw)      First(rw)       Range (rw)
    Lease(rw)           InUse(ro)

00  1000                on              192.168.5.80    9
    15                  0

01  1000                on              192.168.5.90    4
    15                  0

mybrick : ipDhcpTable >
```

When an address is assigned, the BRICK keeps track of the address' availability in the ***ipDhcpInUseTable.***

```
mybrick: ipDhcpTable> ipDhcpInUseTable

inx Address(*ro)        Phys(ro)           Expires(ro)

00  192.168.5.80        8:0:20:19:ef:eb    30/06/97 17:33:21
01  192.168.5.81        8:0:20:a3:b3f:9    30/06/97 17:36:59
02  192.168.5.83        8:0:20:12:4f:9a    30/06/97 18:06:19

mybrick : ipDhcpInUseTable >
```

Note that 192.168.5.82 wasn't assigned above. The BRICK verifies an address is available via ping: ICMP requests/reply before assigning an address from the DHCP tables. Since a reboot results in the loss of the audit trail of assigned addresses this is done to avoid duplicate assigments.

According to DHCP, the server may provide clients with additional information such as netmasks, nameserver and other addresses, etc. The table shown below, lists the types of information the BRICK currently supports.

DHCP Request Tag	Retrieved from:
IP_ADDRESS	***ipDhcpTable*** (amd ***ipDhcpInUseTable***) (the next available address, see above)
SUBNET_MASK	***ipRouteTable***
GATEWAY	***ipRouteTable*** IP address of the interface the request was received on.
BROADCAST_ADDR	***ipRouteTable*** (using default route: ***Dest*** | ~ ***Mask*** = broadcast address)
TIME_SERVER	***biboAdmTimeServer***
NAME_SERVER	***biboAdmNameServer, biboAdmNameServ2,*** ***biboAdmWINS1***or ***biboAdmWIN2***, see below.
DOMAIN_NAME	***biboAdmDomainName***
LOG_SERVER	***biboAdmLogHostTable***

DHCP Request Tag	Retrieved from:
HOST_NAME	***ipDhcpTable*** The client's IP address (IP_ADDRESS) is sent as a text string.

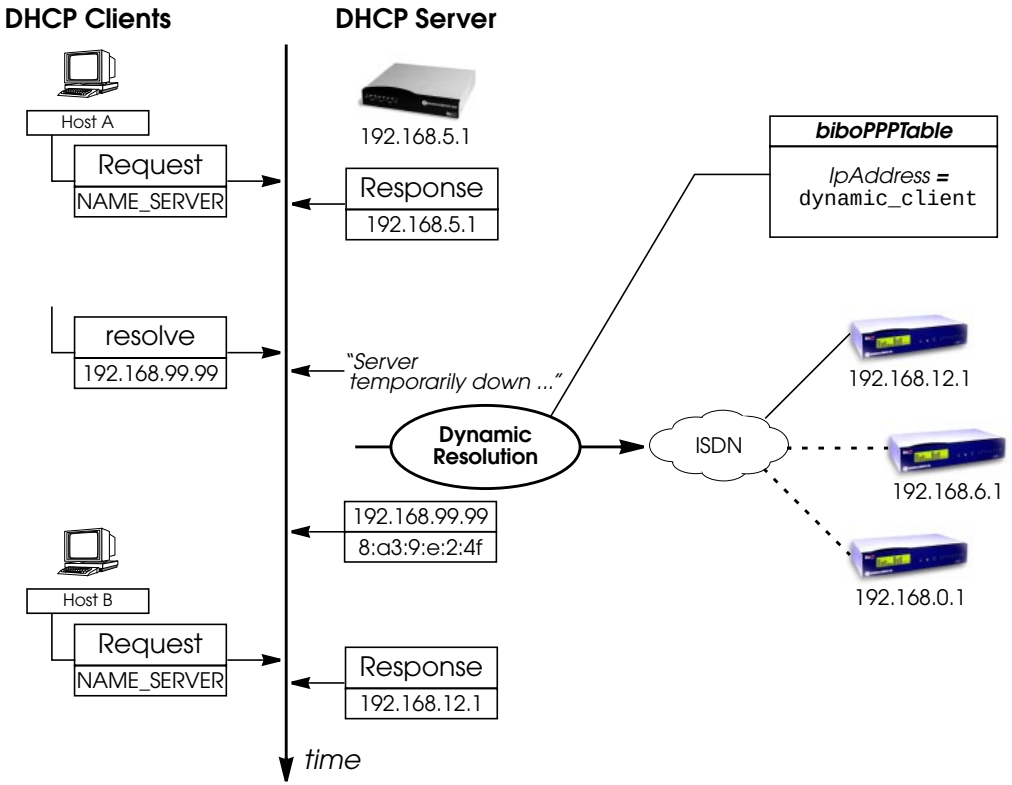
The variables ***biboAdmNameServer, biboAdmNameServ2, biboAdmWNS1*** and ***biboAdmWNS2*** can also be configured in Setup Tool in the **Static** submenu of the **IP** menu. Another way to get the values for these variables is to receive them via PPP like described below.

DNS and WINS (NBNS) Relay

Client messages that include requests for the domain name server or the NetBios server's address are handled as follows.

1. If ***biboAdmNameServer/biboAdmWNS1*** is set (≠ 0.0.0.0) send IP address, otherwise
2. If ***biboAdmNameServ2/biboAdmWNS2*** is set (≠ 0.0.0.0) send IP address, otherwise
3. Send BRICK's IP address as NAME_SERVER and
   attempt to resolve the name server's address dynamically (see below) upon reception of first name resolution request.

Dynamic Name Server Address Resolution



- The BRICK parses the ***biboPPPTable*** for partners that support DNS/WINS negotiation; i.e., ***DNSNegotiation*** is **enabled** and the ***IpAddress*** field is set to **`dynamic_client`** or ***DNSNegotiation*** is set to **dynamic_client**.
- While attempting to configure it's DNS server, DNS requests are answered with a "Server temporarily down" resp. "Server Failure"message.
- WAN partners are only called once.
- After successful DNS Address negotiation, the BRICK can inform subsequent DHCP requests for a name server with its newly configured address. See the section on DNS and WINS (NBNS) Negotiation over PPP, for information on dynamic DNS address negotiation.

- Clients that were given the BRICK's address as name server can't be informed of a new address. For these hosts, the BRICK simply continues relaying resolution requests to the actual DNS/WINS server.

DNS and WINS Addresses over PPP

DNS and WINS (NBNS = NetBios Name Server) negotiation can be configured on a per partner basis and allow to better control how (and from which partners) the BRICK will negotiate DNS and WINS settings.

Each partner can be separately configured so that the BRICK either

- accepts DNS/WINS settings from the partner,
- offers DNS/WINS settings to this partner
- or does not negotiate DNS/WINS settings with the partner.

Together the MIB variables biboPPPIpAddress and biboPPPDNSNegotiation control how DNS/WINS Negotiation is handeled with the respective PPP partner.

biboPPPDNSNegotiation	The type of negotiation to perform with this client. Default value: ***enabled*** Possible values include: ***disabled*** (1), ***enabled*** (2), ***dynamic_client***(3), ***dynamic_server*** (4)
biboPPPIpAddress	The type of IP address for this dial-up partner. Possible values include: ***static*** (1), ***dynamic_server*** (2), ***dynamic_client*** (3)

Th table below illustrates the effect of using this two variables to control DNS and WINS negotiation:

Variable Settings	Negotiation Handling
biboPPPDNSNegotiation: disabled	No negotiation is performed.

Variable Settings	Negotiation Handling
biboPPPDNSNegotiation: enabled *and* biboPPPIpAddress: dynamic_client	DNS resp. WINS addresses are requested from the remote side. Correspondingly the variables ***biboAdmNameServer, biboAdmNameServ2, biboAdmWINS1*** or ***biboAdmWIN2*** are overwritten.
biboPPPDNSNegotiation: enabled *and* biboPPPIpAddress: dynamic_server *or* biboPPPIpAddress: static	DNS resp. WINS addresses are sent to the remote side, when requested, as far as they are configured. The addresses are taken from the variables ***biboAdmNameServer, biboAdmNameServ2, biboAdmWINS1*** or ***biboAdmWIN2***.
biboPPPDNSNegotiation: dynamic_client	DNS resp. WINS addresses are requested from the remote side. Correspondingly the variables ***biboAdmNameServer, biboAdmNameServ2, biboAdmWINS1*** or ***biboAdmWIN2*** are overwritten.
biboPPPDNSNegotiation: dynamic_server	DNS resp. WINS addresses are sent to the remote side, when requested, as far as they are configured. The addresses are taken from the variables ***biboAdmNameServer, biboAdmNameServ2, biboAdmWINS1*** or ***biboAdmWIN2***.

In Setup Tool the values of the variables ***biboPPPDNSNegotiation*** and ***biboPPPIpAddress*** can also be configured:

When you configure a WAN Partner, you will find the item **Dynamic name Server Negotiation** in the **Advanced** submenu of the WAN partner's **IP** menu. This item corresponds to the variable ***biboPPPDNSNegotiation.***

The value of the ***biboPPPIpAddress*** variable is configured via the item **IP Transit Network** in the WAN partner's **IP** menu. Here the settings **yes** and **no** correspond to the variable's value ***static*** and **dynamic client** to *dynamic_client*, the same **dynamic server** to *dynamic_server*.

Dynamic IP Address Assignment

As the name suggests, Dynamic IP Address Assignment is a method used to configure a host's IP address dynamically. It's generally based on a client-server system; clients ask for an address and the server assigns one.

Although it's used for a variety of reasons by different sites, it's primary benefit is that it allows for efficient (and centralized) management of a limited number of IP addresses. Internet service providers commonly use it to assign IP addresses to dial-in hosts at connections time.

> **NOTE**: DHCP can also be used to provide hosts with an IP address (and other information, see: BOOTP and DHCP) but is mainly used for assigning IP addresses to hosts on the BRICK's LAN.

Dynamic IP Address Assignment on the BRICK is used for hosts that connect to the BRICK via ISDN; the BRICK can operate as a Server or as a Client for such hosts. Configuring Server Mode or Client Mode is described below.

Server Mode

It is possible to define separate IP Address Pools for dynamic IP address assigments. For Internet Service Providers (ISP) and other sites with many dial-in accounts, using IP address pools is convenient for defining separate user groups. One might assign "official" addresses from one pool 1 for special accounts, and assign "non-official" addresses from pool 2 for private accounts.

In server mode, the BRICK assigns an IP address to a host (the client) at connection time from the Pool (Pool ID) defined for the respective WAN Partner. When dynamically assigning an IP address to a dial-in client the static IP address respectively the Pool from which the address is retrieved are determined in the following order.

1. **Assigning a Static IP Address**
   When there exists an entry in the ***ipRouteTable*** for the dial-in client, where ***ipRouteMask*** is set to a host route (= ***255.255.255.255***) and ***ipRouteType*** has the value ***direct***, in this case the IP address stored in the variable ***ipRouteDest*** of this routing entry is taken to be assigned for this WAN partner.
   If caller can't be authenticated locally via the MIB, RADIUS server(s) are contacted. If a server authenticates the caller, and there is a User-Record entry BinTec-ipRouteTable="ipRouteMask=255.255.255.255

ipRouteType=direct
ipRouteDest= x"

the IP address stored in the variable ***ipRouteDest*** of this entry is taken to be assigned for this WAN partner.

2. **Assigning an IP Address from an Address Pool**
   When the procedure described under 1.) was not successful, the IP address is assigned from the Pools.
   Once the caller is identified (either inband or outband), the WAN partner's ***biboPPPTable*** entry is compared. If the ***IPAddress*** field = "dynamic_server" AND an address is available from the pool identified by the ***PoolId*** field, then a free address is assigned.
   If caller can't be authenticated locally via the MIB, RADIUS server(s) are contacted. If a server authenticates the callerand there is a User-Record entry BinTec-biboPPPTable="biboPPPIpAddress=dynamic_server", the pool ID is determined from the User-Record entry BinTec-biboPPPTable="biboPPPIp-PoolId=x".

For detailed description of individual system table fields please refer to the BIANCA/BRICK MIB Reference on the accompanying Companion CD or at *BinTec's WWW* site.

Example Configuration of an IP Address Pools via Setup Tool

A. Dial-In Partner without RADIUS

IP → DYNAMIC IP ADDRESS → ADD

First, create/modify a Pool ID to contain IP addresses that will be available for assignment at connect time.

Pool ID	1
Number	10.5.5.5
Number of Consecutive Addresses	5

WAN PARTNER → ADD

Here you'll need to set:

Partner Name	test
Encapsulation	*PPP*
Compression	none
Encryption	none
Calling Line ID	no

Then, in the IP submenu configure the BRICK as a Dynamic IP Address server for this partner.

IP Transit Network	dynamic_server

In the ADVANCED SETTINGS submenu define the Pool ID

IP Address Pool	1

B. Dial-In Partner with RADIUS server

IP → DYNAMIC IP ADDRESS → ADD

Next, modify a Pool ID to contain IP addresses that will be available for assignment at connect time.

Pool ID	2
Number	192.168.80.20
Number of Consecutive Addresses	20

Then you must define the following entry in the User-Record of the RADIUS server:

BinTec-biboPPPTable=”biboPPPIpPoolId=2”

Example Configuration of IP Address Pools via SNMP Shell

A. Dial-In Partner without RADIUS

1. Create an IP address pool in the ***biboPPPIpAssignTable***.

```
brick:> biboPPPIpAssignAddress=10.5.5.5 biboPPPIpAssignPoolId=1 biboPPPIpAssignRange=5
```

2. Set the WAN partner in ***biboPPPTable*** to use Pool ID.
   Assuming entry 4 is the existing WAN partner we want to configure for Dynamic IP address assignment.

```
brick:> biboPPPIpPoolId:4=1 biboPPPIpAddress:4=dynamic_server
```

B. Dial-In Partner with RADIUS server

1. Create an IP Address pool in the ***biboPPPIpAssignTable***.

```
brick:> biboPPPIpAssignAddress=192.168.80.20 biboPPPIpAssignPoolId=2
        biboPPPIpAssignRange=20
```

2. Define the following entry in the User-record of the RADIUS server:

   BinTec-biboPPPTable=”biboPPPIpPoolId=2”

3. Once the caller is authenticated via a RADIUS server a temporary ***biboPPPTable*** entry is generated. The ***PoolId*** field for this entry is determined by the contents of the User-Record discussed above.

Overlapping Address Pools

Although it’s legally possible to define IP address pools that overlap (as shown below) the BRICK will not assign an address twice.

The ***biboIpInUseTable*** is consulted for this purpose. The ***biboIpInUseTable*** shows all IP addresses, which are dynamically assigned to WAN partners or reserved for WAN partners and is continously updated.

Example for overlapping Address Pools:

```
brick: biboPPPIpAssignTable> biboPPPIpAssignTable

inx     Address(*rw) State(-rw) PoolId(rw)     Range(rw)

0       10.5.5.1     unused     0              2
1       10.5.5.2     unused     1              2
2       10.5.5.3     unused     2              2

brick: biboPPPIpAssignTable>
```

With the ***biboPPPIpAssignTable*** shown above, only four IP addresses could actually be used at any given time.

An address pool may be removed from the table at any time by assigning **`delete`** to the respective ***State*** object.

Reserved IP Addresses

In the ***biboIpInUseTable*** all IP addresses currently assigned or reserved to a partner are shown.

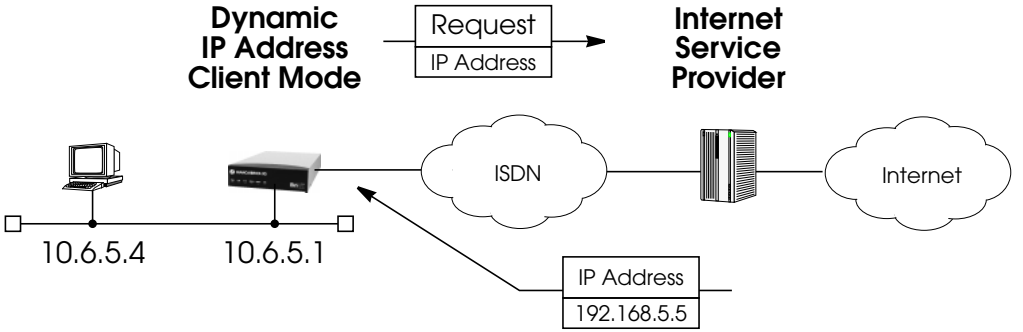
After a disconnect the ***State*** of the entry is set to reserved and the variable ***PPPIPInUseAge*** is reset to 0. From then on it is tried to reserve the IP address for the partner for a maximum of 3600 s.

When within this time the same partner calls again, the BRICK tries to assign the respective address again. The assignement is made via the variable ***PPPIpInUseIdent***.

When a partner dials in and no reserved IP address is available, the next step is to assign a free IP address from the specified pool. When no more free IP address is available from the pool, the oldest of the IP addresses, reserved for other partners, is used.

Client Mode

In client mode, the BRICK can be configured to accept its own IP address at connection time from a dial-up PPP partner at connection time. The BRICK will use this IP address as the local side of the dialup connection as long as the connection is established.



Upon receiving an IP address from the dialup (server) host, the BRICK will automatically create an IP route to allow hosts on the LAN to access networks via the dialup connection. The route is also automatically removed when the connection is closed.

Routing with OSPF

OSPF on the BRICK consists of 11 system tables and the ospfmon application. An overview of the 10 OSPF tables (from the **`ospf`** group) and the ***ipImportTable*** (**`ip`** group) from the SNMP shell are shown below.

OSPF System Tables

- ***ospfGeneralGroup***
  Global settings used by the OSPF protocol including the ***ospfAdminStat*** object (must be enabled to use OSPF).
- ***ospfStatTable***
  Status information about Link-State advertisements and OSPF protocol packets that have been sent or received.
- ***ospfErr***
  Status information about bad OSPF packets (bad checksum, incorrect field values, etc.) that have been received.

- ***ospfAreaTable***
  Identifies OSPF areas the BRICK's interfaces are assigned to and logs statistics for each.
- ***ospfLsdbTable***
  Contains header information from the BRICK's Link State DB.
- ***ospfIfTable***
  Lists all OSPF interfaces, their current state, and settings specific to that OSPF interface.
- ***ospfIfMetricTable***
  Lists the actual metric values being used for each OSPF interface.
- ***ospfNbrTable***
  Lists the neighbor routers that have been identified via then HELLO protocol and their respective OSPF states.
- ***ospfAreaAggregateTable***
  Specifies IP address ranges for route condensation (also called: inter-area route summarization) among areas.
- ***ospfStubAreaTable***
  Generates a default route for Stub Areas.
- ***ipImportTable***
  Specifies how routes from one routing protocol are imported into another routing protocol.

Example OSPF Installation

A typical network installation showing how OSPF could be put to use is shown in the diagram on the following page. Highlights for this setup are shown below. Following the diagram is a Configuration Overview and following that a detailed listing of the configuration steps is povided for each router.

Area 11.0.0.0 (stub area)

- Since the remote LAN in Area 11.0.0.0 is linked to the backbone via an ISDN dialup link this area is config ured as a stub area. This means that external routing in formation advertisements won't flow into this area. The default route for this area is provided by the router BRICK-XL.

- Because OSPF on the BRICK includes support for Demand Circuits (RFC 1793) the dialup link is only opened when changes in routing information must be propagated.

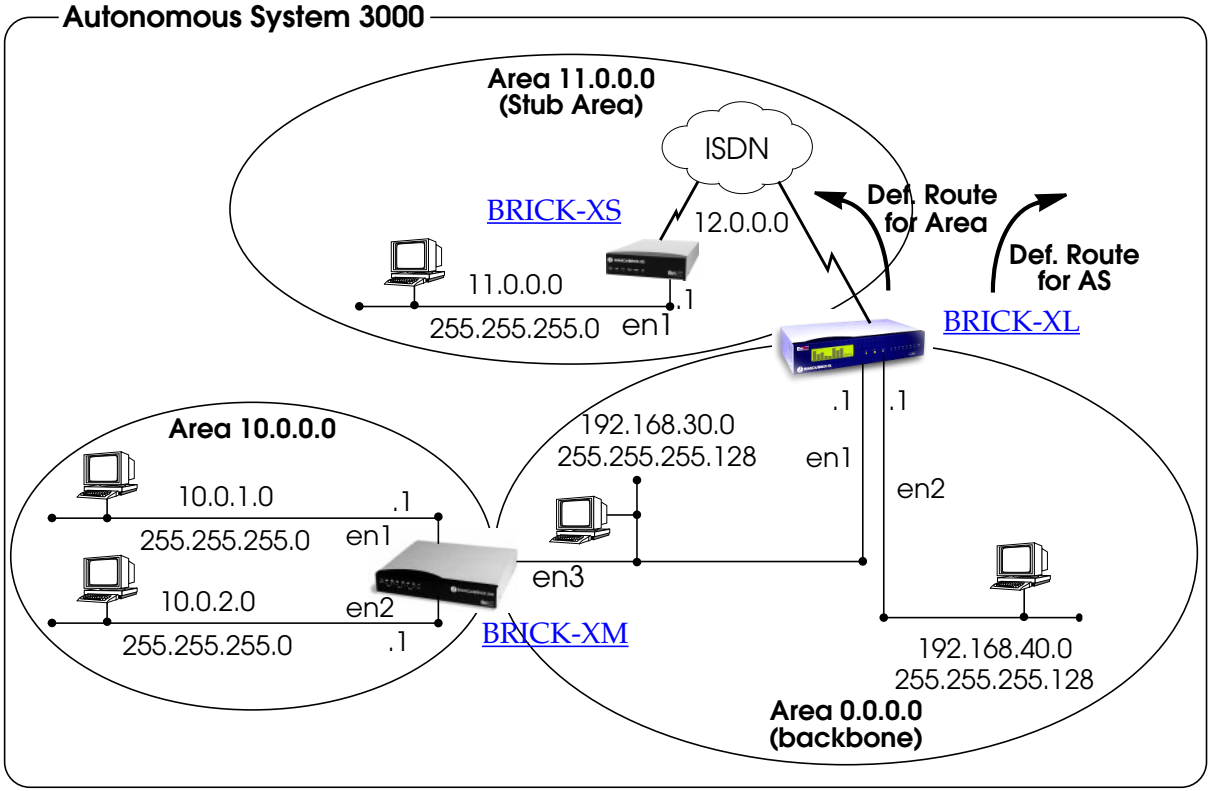
Area 0.0.0.0 (backbone)

- Area 0.0.0.0 is the backbone of the Autonomous System. The router at BRICK-XL will provide the default route for the entire AS and a default route for Area 11.0.0.0.

Area 10.0.0.0

- Area 10.0.0.0 is connected to the backbone via the border router BRICK-XM. Since this is the only link between networks in this area and any external networks (such as the Internet) BRICK-XM will provide Summary Links to routers in other areas. This means that routing information about networks in Area 10.0.0.0 will be combined (or aggregated) into a single advertisement. This

lessens the amount of traffic on the backbone and keeps the size of the link state database for area 0.0.0.0 small.



Configuration Overview

All BRICKs:

1. A valid OSPF license must be installed. This can be added to the *biboAdmLicenseTable* or from Setup Tool's LICENSES → menu.
2. OSPF must be enabled by setting *ospfAdminStat* to **enabled**, or from Setup Tool's IP → OSPF → STATIC SETTINGS → menu.

BRICK-XL Overview (details):

1. Create the dial-up partner interface to BRICK-XS.
2. Have BRICK-XL advertise the default route for the AS.
3. Create the Area entry for Area 11.0.0.0.
4. Assign the new dialup partner interface to Area 11.0.0.0 and set the interface to active.
5. Verify ethernet interfaces en1 and en2 are assigned to Area 0.0.0.0 and set both interfaces to active.

BRICK-XS Overview (details):

1. Create the dial-up partner interface to BRICK-XL.
2. Create the Area entry for Area 11.0.0.0.
3. Assign the ethernet interface (en1) to Area 11.0.0.0 and set the interface to active.
4. Assign the new dial-up interface to Area 0.0.0.0 and set the interface to active.

BRICK-XM Overview (details):

1. Create the Area entry for Area 10.0.0.0.
2. Assign ethernet interfaces en1 and en2 to Area 10.0.0.0 and set both interfaces to active.
3. Verify ethernet interface en3 is assigned to Area 11.0.0.0 and set the interface to active.
4. Create the OSPF aggregate for the LANs attached to en1 and en2 to reduce the routing traffic sent over en3.

Configuration Steps for BRICK-XL

1. Assuming an OSPF license is installed and OSPF has been enabled the partner interface to BRICK-XS should be created. Note that our example uses a transfer network (network 12.0.0.0).
2. Since BRICK-XL should advertise the default route for the AS set this field to yes in IP → OSPF → STATIC SETTINGS →.

```
BIANCA/BRICK-XL Setup Tool                    BinTec Communications AG
(IP) (OSPF) (STATIC): OSPF Static Settings                    BRICK-XL

OSPF                                          enabled
Generate Default Route for the AS             yes

                        SAVE                  CANCEL

Enter IP address (a.b.c.d or resolvable hostname)
```

3. In the IP → OSPF → AREAS → menu create an entry for Area 11.0.0.0. Define this area as a Stub Area and have BRICK-XL generate the default route for this area.

```
BIANCA/BRICK-XL Setup Tool                    BinTec Communications AG
(IP) (OSPF) (AREA) (ADD): Area Configuration                  BRICK-XL

Area ID                                       11.0.0.0

Import external routes                        no
Import summary routes                         no
Create area default route (only ABR)          yes

Area Ranges >

                        SAVE                  CANCEL

Enter IP address (a.b.c.d or resolvable hostname)
```

4. In the IP → OSPF → INTERFACES → menu locate the dialup interface entry created in step 1 and hit enter to edit the settings.

   Set the Admin Status to active and assign it to Area 11.0.0.0 (or the area created in step 3) and select SAVE.

```
BIANCA/BRICK-XL Setup Tool                          BinTec Communications AG
(IP) (OSPF) (INTERFACE): Configure Interface BRICK-XS                BRICK-XL

   Admin Status                  active (propagate routes + run OSPF)
   Area ID                       11.0.0.0

   Metric Determination          auto (ifSpeed)
   Metric (direct routes)        1562

   Authentication Type           none
   Authentication Key

   Import indirect static routes no

                    SAVE                CANCEL

Use (Space) to select
```

   By default, dial-up interfaces are set to passive in the Admin Status field.

5. In IP → OSPF → INTERFACES → menu verify the ethernet interfaces en1 and en2 are assigned to the backbone, (Area 0.0.0.0 which is the default area).

   Set the Admin Status to active and assign it to Area 11.0.0.0 (or the value from step 2) and select SAVE

Configuration Steps for BRICK-XS

1. Assuming an OSPF license is installed and OSPF has been enabled the dial-up partner interface to BRICK-XL should be created. In our example a transfer network (12.0.0.0) is used.
2. In the IP → OSPF → AREAS → menu create Area 11.0.0.0. and define it as a Stub Area.

```
BIANCA/BRICK-XS Setup Tool                    BinTec Communications AG
(IP) (OSPF) (AREA) (ADD): Area Configuration                  BRICK-XS

    Area ID                                   11.0.0.0

    Import external routes                    no
    Import summary routes                     no
    Create area default route (only ABR)      no

    Area Ranges >

                        SAVE                  CANCEL

Enter IP address (a.b.c.d or resolvable hostname)
```

3. In the IP → OSPF → INTERFACES → menu assign the ethernet interface (en1) to Area 11.0.0.0 and make sure the Admin Status is set to active.

```
BIANCA/BRICK-XS Setup Tool                    BinTec Communications AG
(IP) (OSPF) (INTERFACES) Configure Interface en1              BRICK-XS

    Admin Status                              active (propagate routes + run OSPF)
    Area ID                                   11.0.0.0

    Metric Determination                      auto (ifSpeed)
    Metric (direct routes)                    10

    Authentication Type                       none
    Authentication Key

    Import indirect static routes             no

                        SAVE                  CANCEL

Use (Space) to select
```

4. In IP → OSPF → INTERFACES → menu locate the dialup interface (created in step 1) and assign the interface to Area 11.0.0.0 (or the value used in step 2).

   Set the Admin Status for the dialup interface to active and select SAVE.

```
BIANCA/BRICK-XS Setup Tool                    BinTec Communications AG
(IP) (OSPF) (INTERFACES) Configure Interface dialup              BRICK-XS

  Admin Status                 active (propagate routes + run OSPF)
  Area ID                      11.0.0.0

  Metric Determination         auto (ifSpeed)
  Metric (direct routes)       1562

  Authentication Type          none
  Authentication Key

                        SAVE              CANCEL

Use (Space) to select
```

Configuration Steps for BRICK-XM

1. An OSPF license must already be installed and OSPF should been enabled
IP → OSPF → STATIC SETTINGS → menu.

   Then create an area entry for Area 10.0.0.0 in the
IP → OSPF → AREAS → menu.

```
BIANCA/BRICK-XM Setup Tool                    BinTec Communications AG
[IP][OSPF][AREA][ADD]: Area Configuration                     BRICK-XM

    Area ID                          10.0.0.0
    Import external routes           yes

    Area Ranges >

                        SAVE                 CANCEL

Enter IP address (a.b.c.d or resolvable hostname)
```

2. In the IP → OSPF → INTERFACES → menu assign ethernet interfaces en1 and en2 to Area 10.0.0.0 (or the value from the previous step) and set the Admin Status for each interface to active.

```
BIANCA/BRICK-XM Setup Tool                    BinTec Communications AG
(IP) (OSPF) (INTERFACES) Configure Interface en1              BRICK-XM

    Admin Status                     active (propagate routes + run OSPF)
    Area ID                          10.0.0.0

    Metric Determination             auto (ifSpeed)
    Metric (direct routes)           10

    Authentication Type              none
    Authentication Key

    Import indirect static routes    no

                        SAVE                 CANCEL

Use (Space) to select
```

3. Ethernet interface en3 should already be assigned to the backbone, Area 0.0.0.0 which is the default.

   In the IP → OSPF → INTERFACES → menu verify this setting and change the Admin Status to active.

4. Return to the IP → OSPF → AREA → menu and scroll to the Area ID entry for the backbone and hit enter.

   Move to the AREA RANGES → submenu to add an OSPF aggregate for the LANs attached to en1 and en2. The Address and Mask entries shown below will match any routes with a destinations starting with 10, or 10.*.*.*.

```
BIANCA/BRICK-XM Setup Tool                    BinTec Communications AG
(IP) (OSPF) (AREA) (RANGE) (ADD): Configure Address range for Area BRICK-
XM

                    Address              10.0.0.0
                    Mask                 255.0.0.0
                    Advertise Matching   yes

                        SAVE                CANCEL

Enter IP address (a.b.c.d or resolvable hostname)
```

This entry means that BRICK-XM will consolidate multiple routes (routes for destinations in Area 10.0.0.0) into a single link state advertisement.

This will effectively reduce the amount of traffic sent over the backbone as will help keep the size of the link state database and routing tables for routers in other areas to a minimum.

Configuring OSPF Virtual Links

A virtual interface must be defined on each of the ABRs by creating an entry in the ***ospfVirtIfTable***. This is done by setting the ***ospfVirtIfNeighbor*** and ***ospfVirtIfAreaID*** objects.

ospfVirtIfNeighbor should be set to the Router ID of the Area Border Router at the oher end of the virtual link.

ospfVirtIfAreaID should be set to the area ID of the transit area.

The virtual link in the diagram here would be configured on Brick-A as follows.

```
BRICK-A:system> ospfVirtIfTable

inx Areald(*rw)          Neighbor(*rw)       TransitDelay(rw)
    RetransInterval(rw)  HelloInterval(rw)   RtrDeadInterval(rw)
    State(ro)            Events(ro)          AuthKey(rw)
    Status(-rw)          AuthType(rw)

BRICK-A:ospdVirtIfTable> AreaID=10.0.0.0 Neighbor=10.0.1.2
```

This creates a new OSPF virtual interface (on BRICK-A) that links two parts of the backbone via the transit area 10.0.0.0. The respective interface would be created on BRICK-B using almost the same command (***ospfVirtIfAreaID***=10.0.0.0 ***ospfVirtIfNeighbor***=10.0.1.1)

Remember that the area being used as the transit area must already be defined in the ***ospfAreaTable***.

Controlling Link State Database Overflow

Sites with large (or complicated) network installations that are running OSPF may notice the Link State Database (LSDB) becoming large. Most often this is the case where external routes are being imported as external advertisements.

One way to minimize the size of the LSDB (on the BRICK) is to use the ***ospfExtLsdbLimit*** variable. This object defines the maximum number of external LSAs to store in the database (the local copy).

Once the limit is reached the BRICK goes into Overflow State. In Overflow State two things happen:

1. The BRICK begins to flush all external advertisements generated locally.

2. The BRICK ignores all new external advertisements.

NOTE: The maximum size of the LSDB must be the same for all OSPF routers in the domain for this feature to perform efficiently.

By default the BRICK remains in overflow state but can optionally be configured to leave overflow state (and continue to process new external LSAs) automatically after a time period. The ***ospfExtOverflowInterval*** variable defines the number of seconds to wait before leaving overflow state automatically. The default is 0 seconds (i.e., stay in overflow state). After waiting ***ospfExtOverflowInterval*** seconds the number of external LSAs in the LSDB is compared to the ***ospfExtLsdbLimit***. If there is room in the database for new LSAs the BRICK then leaves overflow state; otherwise another time interval is waited.

The diagram shown below attempts to illustrate the behavior of database overflow control using the ***ospfExtLsdbLimit*** and ***ospfExtOverflowInterval*** variables.

Enabling Demand Circuit Support

Demand Circuit support for dial-up partner interfaces is enabled by default when an existing interface is enabled for OSPF (AdminStatus is set to active). Support can be manually controlled by setting the interface's ***IfDemand*** object (***ospfIfTable***) to "true" or "false". When set to false, the state of this interface is always up.

Setting this variable to true for one side of the connection is sufficient (that is, as long as OSPF has been enabled on both sides, i.e., ***ipExtIfOspf***=active) if both sides support RFC 1793.

Note: Until a neighbour router has been identified HELLO packets are periodically transmitted (default, ***ospfIfPollInterval*** = 120 seconds) over the interface. This results in the link being opened. Once the LSDB has been synchronised, the HELLO protocol is then suppressed.

Import - Export of Routing Information

When different routing protocols are used within the same domain it is sometimes useful to be able to exchange (import or export) routing information between these protocols.

Using the ***ipImportTable*** routing information generated by one protocol (***ipImportSrcProto***) can be imported or exported to another protocol (***ipImportDstProto***).

Currently the following ***SrcProto↔DstProto*** combinations are possible.

ipImportSrcProto \ ipImportDstProto	rip	ospf
default_route		3[1]
direct		
static		3[2]
rip	-	
ospf	3[3]	-

1. ***ipImportSrcProto***=default_route ***ipImportDstProto***=ospf
   This entry forces an external Link State Advertisement to be generated that defines a default route for the Autonomous System.
2. ***ipImportSrcProto***=static ***ipImportDstProto***=ospf
   With this entry statically configured indirect routes will be propagated via OSPF as external LSAs.
3. ***ipImportSrcProto***=ospf ***ipImportDstProto***=rip
   With this entry, all routes learned via OSPF are imported to RIP. If an OSPF route changes the import to RIP will triggered an immediate broadcast of the entire routing table.

The remaining fields of ***ipImportTable*** allow for further control of how (and what) routing information is imported.

- ***ipImportMetric1***
  The metric in the context of the destination protocol the imported routes should get. If sset to -1 these routes get a protocol specific default metric.

- ***ipImportType***
  This object might define protocol specific properties of the imported routes in the context of the destination protocol.
- ***ipImportAddr***
  Specifies (together with ***ipImportMask***) the range of IP addresses for which the table entry should be valid. The entry is valid if the destination IP address of the route lies in the range specified by both objects. If both objects are set to 0.0.0.0, the table entry will be valid for destination.
- ***ipImportMask***
  Together with ***ipImportAddr*** specifies the range of IP addresses for which the table entry should be valid. For example, if Addr=X.X.0.0 and Mask=255.255.0.0 then addresses X.X.0.0 through X.X.255.255 are valid.
- ***ipImportEffect***
  Defines the effect of this entry. If set to “import”, importation from ***SrcProto*** to ***DstProto*** takes place. If set to “doNotImport” importation is prevented.
- ***ipImportIfIndex***
  Specifies the interface index of the interface for which the entry should be valid. If set to 0 the entry is valid for all interfaces.

Advanced IP Features

Several advanced IP features which are described below are available via settings in the ***ipExtIfTable*** shown below.

```
mybrick: admin> ipExtIfTable

inx Index(*ro)          RipSend(rw)           RipReceive(rw)
    ProxyArp(rw)        Nat(rw)               NatRmvFin(rw)
    NatTcpTimeout(rw)   NatOtherTimeout(rw)   NatOutXlat(rw)
    Accounting(rw)      TcpSpoofing(rw)

mybrick: ipExtIfTable >
```

IP Session Accounting

In many cases, the logging of each individual TCP/IP session is desirable. IP accounting can be turned on for an interface by setting the interface's *Accounting* object in the ***ipExtIfTable*** to "on". Then, for each TCP, UDP or ICMP session that is routed over the interface, an entry in the ***ipSessionTable*** is created. Since this table is updated dynamically, viewing this table allows one to monitor all active sessions.

Once a session is terminated, either by disconnection of the TCP session or timeout, an accounting record is written to the ***biboAdmSyslogTable*** (***Subject***=acct, ***Level***=info). Records can also be sent to remote log hosts using the syslog protocol (see the section Logging with Remote LogHosts, in Chapter 7, System Administration on the BRICK.

NOTE:

Logging with the Syslog protocol is unreliable. In seldom cases accounting records may get lost. Generally, it's best to configure log hosts that are on the same LAN segment as the BRICK, use caution when configuring hosts that can only be reached via ISDN.

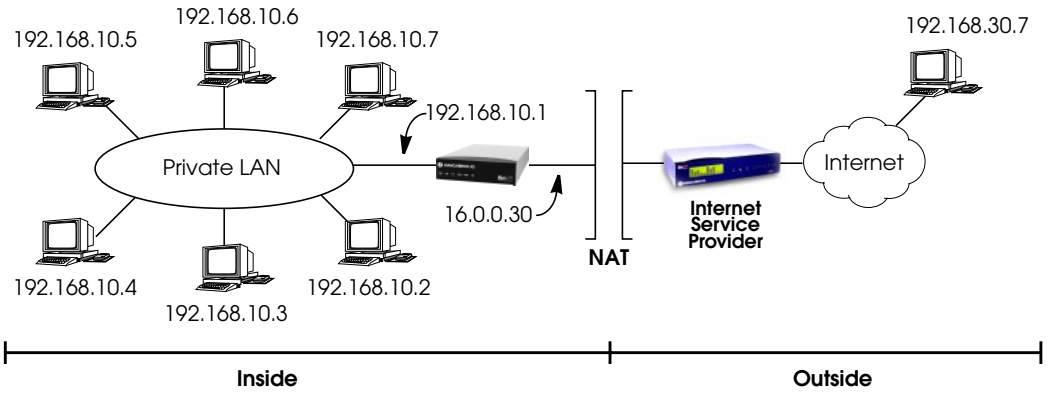
Network Address Translation

NAT (Network Address Translation) allows the BRICK to hide a complete LAN behind one IP address and may be useful in different installations where:

- Security is an issue.
  (controlling access to a limited number of hosts)
- The number of available IP addresses is limited.
- Monitoring of incoming connections is desired.

The BRICK performs NAT by keeping track of all TCP/IP sessions and manipulating all incoming/outgoing IP packets to reflect different source and destination addresses.

In the diagram shown below, the BRICK's ISDN interface can be configured for NAT. All hosts on the Private LAN at 192.168.10.0 can still access external networks, but will appear to external hosts as the same host (16.0.0.30). Connections initiated from outside the private LAN can access local hosts only after local hosts have been explicitly configured (on the BRICK) to accept connections from external hosts. Finally, it is also possible to specify the IP address translation and remote address(es) for outgoing sessions.



Enabling NAT

Step 1

First, a route for the externally visible IP address is needed. For our example shown above, we would create a route to use our ISDN interface with:

```
mybrick: admin> ipRouteIfIndex=10001 ipRouteDest=16.0.0.30 ipRouteNexthop=16.0.0.30
                ipRouteMask=255.255.255.255 ipRouteType=direct

02: ipRouteIfIndex.16.0.0.30.2( rw):      10001
02: ipRouteDest.16.0.0.30.2( rw):         16.0.0.30
02: ipRouteNextHop.16.0.0.30.2( rw):      16.0.0.30
02: ipRouteMask.16.0.0.30.2( rw):         255.255.255.255
02: ipRouteType.16.0.0.30.2(-rw):         direct

mybrick: ipRouteTable >
```

NOTE: This example NAT configuration assumes that the LAN is properly configured and that the appropriate routes are present to allow hosts to connect to external networks.

Step 2

Next, enable the interface the BRICK should perform NAT for; this is “10001” for our example from step 1. Locate the appropriate table entry in the ***ipExtIfTable***, and set ***Nat*** to “on”.

```
mybrick: ipExtIfTable> ipExtIfTable

inx Index(*ro)              RipSend(rw)             RipReceive(rw)
    ProxyArp(rw)            Nat(rw)                 NatRmvFin(rw)
    NatTcpTimeout(rw)       NatOtherTimeout(rw)     NatOutXlat(rw)
    Accounting(rw)          TcpSpoofing(rw)         AccessAction(rw)
    AccessReport            Ospf(rw)                OspfMetric(rw)
    TcpCKsum(rw)            BackRtVerify(rw)        RuleIndex(rw)
    Authentication(rw)      RouteAnnounce(rw)
...
02  10001                   ripV1                   both
    off                     off                     yes
    3600                    30                      on
    off                     off                     ignore
    info                    active                  auto
    check                   off                     0
    off                     strict                  3600
    60                      up_dormant
...
mybrick: admin> ipExtIfNat:02=on
02: ipExtIfNat.10001.2( rw):      on

mybrick: ipExtIfTable >
```

At this point, all hosts on the LAN are inaccessible from external networks but can continue to establish external connections at will.

Allowing Incoming Connections

To allow network connections to hosts on the private LAN (from external networks/hosts), explicit permission must be configured in the ***ipNatPresetTable***. Packets arriving from external networks and addressed to different (official) addresses can be linked to various (internal) addresses. Each entry in this table defines a specific port on a specific host that can be accessed from outside the private LAN.

For each entry, the following variables must be defined:

Ifindex	The *Ifindex* NAT is being performed on,
IntAddr	The host that is to be accessed
Protocol	The protocol (TCP, UDP, ICMP) to allow.
IntPort	The port on the specified host to allow (ftp, telnet, nntp, etc). Required only if the Protocol uses ports (TCP and UDP). –

In the following example, several servers can be entered for the same service in the ***ipNatPresetTable.*** In oder that each of these can be reached with a different external address, the external addresses must be entered in the ***ipNatPresetTable*** and the external mask set to ***255.255.255.255.***

```
inx IfIndex(*rw)      Protocol(*-rw)    RemoteAddr(rw)    RemoteMask(rw)
    ExtAddr(rw)       ExtMask(rw)       ExtPort(*rw)      ExtPortRange(rw)
    IntAddr(rw)       IntPort(rw)

00  10001             tcp               0.0.0.0           0.0.0.0
    192.1.1.1         255.255.255.255   80                -1
    10.1.1.1          -1

01  10001             tcp               0.0.0.0           0.0.0.0
    192.1.1.2         255.255.255.255   80                -1
    10.1.1.2          -1

02  10001             tcp               0.0.0.0           0.0.0.0
    192.1.1.3         255.255.255.255   80                -1
    10.1.1.3

mybrick: ipNatPresetTable >
```

We can use the special internal address 0.0.0.0 to allow access to all hosts on our LAN. The entry below would be used to allow all ICMP packets to enter the private LAN.

```
mybrick: admin>ipNatPrIfIndex=10001 ipNatPrIntAddr=0.0.0.0 ipNatPrProtocol=icmp ipNat-
PrExtPort=-1

inx IfIndex(*rw)      Protocol(*-rw)   RemoteAddr(rw)   RemoteMask(rw)
    ExtAddr(rw)       ExtMask(rw)      ExtPort(*rw)     ExtPortRange(rw)
    IntAddr(rw)       IntPort(rw)

1   0001              icmp             0.0.0.0          0.0.0.0
    0.0.0.0           0.0.0.0          -1               -1
    0.0.0.0           -1
mybrick: ipNatPresetTable >
```

Mapping Addresses for Outgoing Traffic

The *ipNatPresetTable*, however, only controls incoming traffic initiated outside the LAN, allowing access to all or just to specified internal hosts.

In order to map outgoing traffic, i.e. internal addresses initiated within the LAN, to specified and different, external IP addresses, and to specify the remote address(es) packets with these NAT addresses should be sent to, a further table, the ***ipNatOutTable***, is available. In the following example, packets from the hosts with the internal addresses 10.1.1.1 to 10.1.1.3 are to be sent with NAT with the external IP addresses 192.1.1.1. to 192.1.1.3.

```
inx IfIndex(*rw)          Protocol(*-rw)   RemoteAddr(rw)
    RemoteMask(rw)        ExtAddr(rw)      RemotePort(rw)
    RemotePortRange(rw)IntAddr(rw)         IntMask(rw)

00  10001                 any              0.0.0.0
    0.0.0.0               192.1.1.1        -1
    -1                    10.1.1.1         255.255.255.255

01  10001                 any              0.0.0.0
    0.0.0.0               192.1.1.2        -1
    -1                    10.1.1.2         255.255.255.255

02  10001                 any              0.0.0.0
    0.0.0.0               192.1.1.3        -1
    -1                    10.1.1.3         255.255.255.255

mybrick: ipNatOutTable >
```

If no matching entry is found, the IP address is set to the IP address defined on the interface configured for NAT. If a matching entry is found, the source IP address of outgoing IP packets is set to the value of ***ipNatOutExtAddr***. This table is only used if the outgoing address translation is activated (if ***ipExtIfNatOutXlat*** from the ***ipExtIfTable*** is set to on).

Entries in the table are created and removed manually by network management.

The ***ipNatOutTable*** consists of the following variables:

ipNatOutIfIndex This variable specifies the interface index, for which the table entry shall be valid. If set to 0, the entry will be valid for all interfaces configured to use NAT.

ipNatOutProtocol Possible values: icmp (1), tcp (6), udp (17), any (255), delete (256)
This variable specifies the protocol, for which the table entry shall be valid.
Default value: any

ipNatOutRemoteAddr Together with ***ipNatOutRemoteMask,*** this variable specifies the set of target IP addresses for which the table entry is valid. If both variables are set to 0.0.0.0, the table entry will be valid for any target IP address.

ipNatOutRemoteMask Together with ***ipNatOutRemoteAddr,*** this variable specifies the set of target IP addresses for which the table entry is valid. If both variables are set to 0.0.0.0, the table entry rn_wa_511tx.backupwill be valid for any target IP address.

ipNatOutExtAddr This variable specifies the external IP address to which the internal IP address is mapped.

ipNatOutRemotePort Together with ***ipNatOutRemotePortRange***, this variable specifies the range of port numbers for outgoing calls, for which the table entry shall be valid. If both variables are set to -1, the entry is valid for all port numbers. If ***ipNatOutPortRange*** is set to -1, the entry is only valid, when the port number of an outgoing call is equal to ***ipNatOutRemotePort***. Otherwise, the entry is valid, if the called port number is in the range RemotePort .. RemotePortRange.
Default value: -1 Possible values: –1..65535

ipNatOutRemotePortRange Together with ***ipNatOutRemotePort,*** this variable specifies the range of portnumbers for outgoing calls, for which the table entry shall be valid. If both variables are set to –1, the entry is valid for all portnumbers. If ***ipNatOutPortRange*** is set to -1, the entry is only valid, when the portnumber of an outgoing call is equal to ***ipNatOutRemotePort***. Otherwise, the entry is valid, if the called portnumber is in the range RemotePort .. RemotePortRange.
Default value: -1 Possible values: –1..65535)

ipNatOutIntAddr Together with ***ipNatOutIntMask***, this variable specifies the internal host's IP address for outgoing calls matching the table entry. If both variables are set to 0.0.0.0, the table entry will be valid for any source IP address.

ipNatOutIntMask Together with ***ipNatOutIntAddr,*** this variable specifies the internal host's IP address for outgoing calls matching the table entry. If both variables are setto 0.0.0.0, the table entry will be valid for any source IP address.

Session Monitoring

While NAT is operating, you can see a list of established connections in the ***ipNatTable***. This table changes dynamically as sessions from local hosts are opened/closed. A session may be either a tcp connection, a udp connection or an icmp connection with

icmp-echo messages (ping). A valid session is either an outgoing session or an incoming session specified in the ***ipNatPresetTable***. An example ***ipNatTable*** for our installation might look as follows.

```
mybrick: admin>ipNatTable

inx IfIndex(*ro)        Protocol(*ro)     IntAddr(*ro)      IntPort(*ro)
    ExtAddr(ro)         ExtPort(ro)       RemoteAddr(ro)    RemotePort(ro)
    Direction(ro)       Age(ro)

00  10001               tcp               192.168.10.5      21
    16.0.0.30           456               192.168.19.19     80
    incoming            0 00:01:28.00

01  10001               tcp               192.168.10.2      80
    16.0.0.30           456               10.0.70.123       1605
    outgoing            0 00:18:08:40

02  10001               tcp               192.168.10.3      23
    16.0.0.30           456               192.168.55.10     77
    outgoing            0 00:00:05.00

mybrick: ipNatTable >
```

Here we see that three hosts have active sessions.

Entry 00 shows:

The host at 192.168.19.19 has been connected to the FTP service, port 21 on 192.168.10.5.

Entry 01 shows:

Our local host at 192.168.10.2 has an HTTP connection (port 80) open with the host at 10.0.70.123.

Entry 02 shows:

Our local host at 192.168.10.3 has a telnet session (port 23) opened with the host at 192.168.55.10.

The ***Age*** field specifies the period of time since the last packet was sent or received for this session.

Proxies

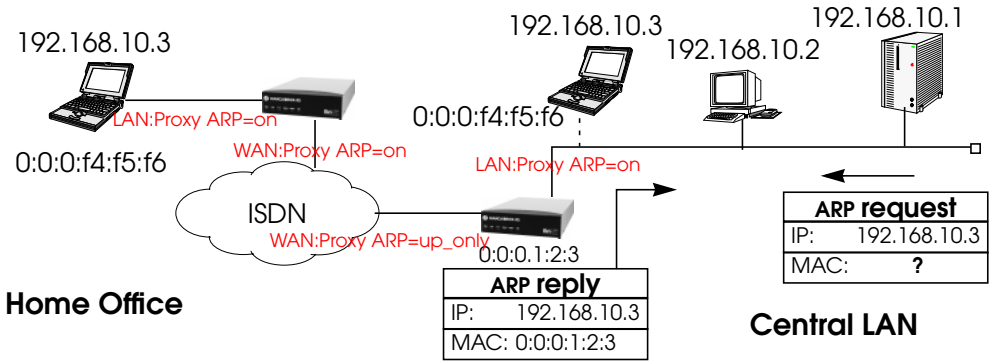
In some cases NAT will not work when port numbers and/or IP addresses are transmitted in the data part of a TCP or UDP session. This is the case for some of the standard Internet Protocols (IP). To allow these protocols to work with NAT, so-called "proxies" have been implemented within the NAT software.

These proxies know how IP addresses and port numbers are transmitted within the data-portion of a connection. The proxy tracks the data sent/received, detects the addresses/port numbers used, and translates the information according to the NAT translation software. Currently, internal proxies have been implemented for the following services:

- FTP
- IRC
- Real Audio
- VDOLive Audio
- rlogin
- RCP
- rsh
- VDOLive Video

Proxy ARP

ARP (Address Resolution Protocol) is a technique used to map an IP address to a physical network address, or MAC address. Normally, ARP requests for the hardware address of a particular IP address are answered by the station the IP address is assigned to. With proxy ARP, the request can be alternatively answered by the BRICK. This is useful when a host on your network is connected via an ISDN line.



Our example above shows a setting, where a laptop is used in the Home Office and is connected to the LAN via ISDN, but may also be connected to the LAN directly. In this example ARP requests from the LAN for the laptop's physical address, are an-

swered by the BRICK, as long as the laptop is connected via ISDN. When the laptop is connected to the LAN, it answers ARP requests itsself.

To activate Proxy ARP it must be turned on for the LAN interface and the destination WAN interface, via which the requested IP address would be routed. The Proxy ARP settings for the WAN interface work in dependence of the operation state of the respective interface, when turned on (***on*** or ***up_only***). IP datagrams from the LAN destined for these hosts are sent directly to the BRICK and are forwarded to the real host. The benefit of proxy ARP is that no routing entries need to be made for such hosts.

For the LAN interface the variable ***ipExtIfProxyArp*** (***ipExtIfTable***) can receive the values off and on:

- ***off***
  Proxy ARP is turned off, which is the default value.
- ***on***
  Proxy ARP is turned on.

In Setup Tool Proxy ARP for the LAN can be configured in the Advanced Settings for the LAN interface.

For the WAN interface the configuration of the variable ***ipExtIfProxyArp*** (***ipExtIfTable***) differs. When proxy ARP is turned on, ARP requests are answered in dependence of the ***ifOperStatus*** (***ifTable***) of the interface, via which the requested host can be reached. Possible values are ***off***, ***on*** and ***up_only***.

Values for ***ipExtIfProxyArp*** on the WAN interface:

- ***off***
  Proxy ARP is turned off, which is the default value.
- ***on***
  The request is only answered, when the WAN interface has the ***ifOperStatus*** ***up*** or ***dormant***. When the interface was in the state ***dormant***, a connection is setup after the ARP request.
- ***up_only***
  The request is only answered, when the WAN interface has the ***ifOperStatus*** ***up***. This value makes sense, when ARP requests should only be answered in case there is already an existing connection to the requested host.

In Setup Tool Proxy ARP for the WAN interface can be configured in the WAN Partner menu for the respective host in the Advanced Settings of the IP submenu.

The requirements for an answer to a ARP request from the LAN by the BRICK are that the destination address would be routed to a different but the LAN interface and that on both interfaces (LAN and destination WAN interface) proxy ARP is turned on (***on*** for the LAN interface and ***on*** or ***up_only*** for the respective WAN interface). Beyond that the ***ifOperStatus*** of the WAN interface must have the demanded state.

When you want to use Proxy ARP on a RADIUS interface, the variable ***ipExtIfProxyArp*** must be set via the BinTec-specific RADIUS attributes. On using BinTec-specific RADIUS attributes see the Extended Feature Reference available via the BinTec FTP server at http://www.bintec.de.

NOTE: Proxy ARP may cause problems on systems that check for security violations where two IP addresses map to the same physical address.

RIP Options

RIP (Routing Information Protocol) is used by IP routers to learn of new IP routes (see RIP for a brief description). To enable the RIP on the BRICK the ***biboAdmRipUdpPort*** field must be set to 520. This is the default setting and specifies the UDP-port to exchange RIP messages over. RIP can be disbled completely by assigning UDP port 0 to this variable.

The BRICK supports both versions 1 and version 2 of RIP. Using the ***RipSend*** and ***RipReceive*** variables in the ***ipExtIfTable***, the BRICK can be configured to separately send/receive either version, both versions or no RIP packets over selected interfaces. ***RipReceive*** defines the types of RIP packets that are accepted (will use for dynamically learning of new routes) over the interface.

ipExtIfRipSend can be assigned the values:

`ripV1`	Send only RIP V1 packets.
`ripV2`	Send only RIP V2 packets.
`both`	Send a RIP V1 packet, and a V2 packet.
`none`	Do not send RIP packets.

ipExtIfRipReceive can be assigned the values:

ripV	Accept only RIP V1 packets.
ripV2	Accept only RIP V2 packets.
both	Accept both versions.
none	Ignore RIP packets.

Back Route Verify

ipExtIfBackRtVerify

This variable activates a check for incoming packets. If set to on, incoming packets are only accepted if return packets sent back to their source IP address would be sent over the same interface. Otherwise, the packets are silently dropped. This prevents packets being passed from untrusted interfaces to this interface.
Possible values: off (1), on (2) Default value: off

8

Chapter Eight

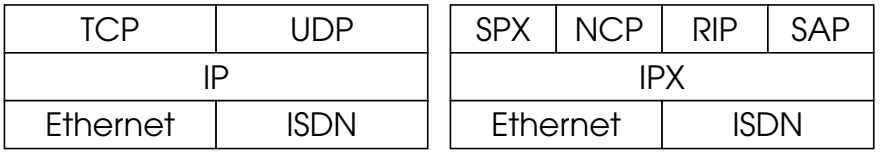
CONFIGURING THE BRICK AS AN IPX ROUTER

What's Covered?

- Introduction to IPX
  - IPX Stations: Servers and Clients
  - IPX Networks: Network Numbers and Addresses
- Configuring IPX Routing
  - Adding Routes and Services
  - Learning Routes and Services
  - Filtering IPX Packets

Introduction to IPX

IPX (Internetwork Packet exchange) is a Network Layer protocol, similar to IP in TCP/IP. An IPX network allows DOS/Windows PCs to share networked services and devices. Services are provided by special PCs which are assigned the duties of, for example, a file or print server.



TCP/IP Networks IPX Networks

IPX (Internetwork Packet exchange) is a connectionless service used to transmit data.

SPX (Sequenced Packet Exchange) is a connection-oriented service used to monitor connections between stations (e.g., a connection to a print service).

Using RIP and SAP routing and service information is periodically exchanged between IPX routers and servers on the network using the RIP and SAP (Service Advertising Protocol) packets.

IPX Stations: Servers and Clients

In an IPX network, stations on the network are classified as either a client or server; and have different characteristics.

Servers

1. Provide special services, (e.g., remote file access, printing, databank access, etc.) to clients.
2. Have a unique name.
3. Can communicate with both servers and clients.

Clients

1. Use the services provided by server stations.
2. Do NOT have unique names.

3. Can ONLY communicate with servers.

IPX Networks: Network Numbers and Addresses

In an IPX network, a network address consists of:

4 byte Network Number
6 byte Node Number
2 byte Socket Number

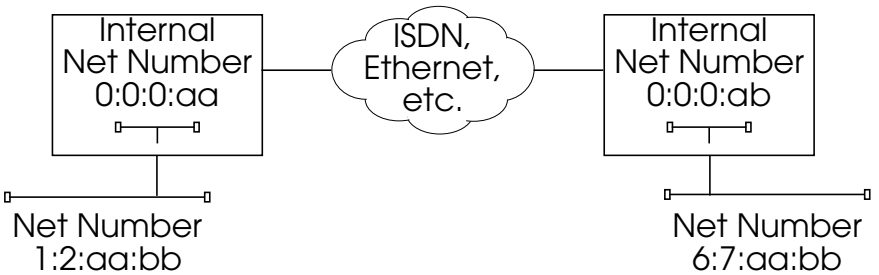
In contrast to IP, where hosts are assigned addresses statically, clients are assigned the Network Number portion of their address dynamically. Servers, on the other hand, have their complete address assigned statically.

Initially, a client asks for its network number by broadcasting a request. A server or router on the network will answer the request with the correct network number. The client then uses the Network Number (received from the server) and its Node Number (normally the MAC address is used), to establish a connection to a server.

Internal Network Numbers

Since IPX uses each stations MAC address for its network address, stations with more than one interface to the network can be reached at different addresses. This can be a problem for a server that advertises services or an IPX router that links multiple IPX LANs.

To get around this problem, servers and IPX routers are assigned Internal Network Numbers. The respective server or router is the only station on this network. By sending RIP packets, routers and servers can inform other stations on the network.



Configuring IPX Routing

Adding Routes and Services

Routes and services the BRICK knows of are learned using the RIP and SAP protocols. This information often changes dynamically. Additional routes and services can be set statically, using the ***ipxStaticRouteTable*** and ***ipxStaticServTable***.

Adding Static Routes

To create a static route to a server the you will need to know the server's internal network number, its name, and the interface the connection should use. The following commands could be used to add a static route to the file server "PHOENIX" which has the internal network number of 0:2:2:2, the route will use the dialup1 interface (***ifIndex*** 10001).

```
mybrick : system > ipxStaticRouteTable

inx SysInstance(*rw)     CircIndex(*rw)      NetNum(*rw)         ExistState(-rw)
    Ticks(rw)            HopCount(rw)

mybrick : ipxStaticRouteTable > ipxStaticServSysInstance=0 ipxStaticRouteCircIndex=10001
                ipxStaticRouteNetNum=0:2:2:2
00: ipxStaticRouteSysInstance.0.10001.0.2.2.2( rw):      0
00: ipxStaticRouteCircIndex.0.10001.0.2.2.2( rw):        10001
00: ipxStaticRouteNetNum.0.10001.0.2.2.2( rw):           0:2:2:2

mybrick : ipxStaticRouteTable > ipxStaticRouteTable

inx SysInstance(*rw)     CircIndex(*rw)      NetNum(*rw)         ExistState(-rw)
    Ticks(rw)            HopCount(rw)

 00 0                     10001               0:2:2:2             on
    0                     0

mybrick : ipxStaticRouteTable >
```

Adding Static Services

A service can also be added statically using the ***ipxStaticServTable***. For services, you will also need to know:

- The socket number
- The type of service the server provides
- The server's Node Number.

Note:

For each ***ipxStaticServNetNum***, the BRICK needs to have a route to the server in ***ipxStaticRouteNetNum***.

The following could be used to add a static service for "PHOENIX" from the previous section.

```
mybrick : ipxStaticRouteTable > ipxStaticServTable

inx SysInstance(*rw)        CircIndex(*rw)        Name(*rw)        Type(*rw)
    ExistState(-rw)         NetNum(rw)            Node(rw)         Socket(rw)
    HopCount(rw)

mybrick:ipxStaticServTable> SysInstance=0 CircIndex=10001 Name=PHOENIX Type=0:4
                            NetNum=0:2:2:2 Node=0:0:0:0:0:1 Socket=4:51

00: ipxStaticServSysInstance.0.10001.7.80.72.79.69.78.73.88.0.4( rw):   0
00: ipxStaticServCircIndex.0.10001.7.80.72.79.69.78.73.88.0.4( rw):     10001
00: ipxStaticServName.0.10001.7.80.72.79.69.78.73.88.0.4( rw):          "PHOENIX"
00: ipxStaticServType.0.10001.7.80.72.79.69.78.73.88.0.4( rw):          0:4
00: ipxStaticServNetNum.0.10001.7.80.72.79.69.78.73.88.0.4( rw):        0:2:2:2
00: ipxStaticServNode.0.10001.7.80.72.79.69.78.73.88.0.4( rw):          0:0:0:0:0:1
00: ipxStaticServSocket.0.10001.7.80.72.79.69.78.73.88.0.4( rw):        4:51

mybrick:ipxStaticServTable> ipxStatisServTable

inx SysInstance(*rw)        CircIndex(*rw)        Name(*rw)        Type(*rw)
    ExistState(-rw)         NetNum(rw)            Node(rw)         Socket(rw)
    HopCount(rw)

 00 0                       10001                 "PHOENIX"            0:4
    on                      0:2:2:2               0:0:0:0:0:1          4:51
    0
mybrick:ipxStaticServTable>
```

Learning Routes and Services

Adding static routes and services for IPX network that change often, or have many servers can be demanding. You can allow the BRICK to learn of routes and services using RIP and SAP and then have the BRICK move all learned information to the Static tables. This is done as follows:

1. Enable RIP/SAP for the PPP interface.
2. Wait until the desired routes and services appear in the *ipxDestTable* and *ipxDestServTable*.
3. Set *ipxAdminLearnStatics* to **both**.
4. Disable RIP and SAP for the PPP interface.

The result of this is that all routes and services learned from PPP interfaces are copied appended from *ipxDestTable* and *ipxDestServTable* to the *ipxStaticRouteTable* and *ipxStaticServTable*.

Note:

Each time the BRICK is allowed to learn statics, the learned information is appended to the Static tables. This may result in duplicate static entries.

Filtering IPX Packets

An important characteristic of IPX networks is the periodic sending of IPX packets between communicating stations over the network. For LAN traffic this is acceptable, but when connecting IPX LANs over ISDN, the amount of RIP and SAP traffic can lead to long (or often) connection times. In addition to the spoofing mechanism IPX traffic can be filtered using the *ipxAllowTable* and *ipxDenyTable*.

For example, serialization packets could be filtered with the following.

```
mybrick : ipxStaticRouteTable > ipxDenyTable

inx PktTypeMode(*-rw)        PktType(rw)          DstIfStatus(rw)
    DstNetMode(rw)           DstNet(rw)           DstNodeMode(rw)
    DstNode(rw)              DstSockMode(rw)      DstSock(rw)
    SrcIfIndexMode(*rw)      SrcIfIndex(rw)       SrcNetMode(rw)
    SrcNet(rw)               SrcNodeMode(rw)      SrcNode(rw)
    SrcSockMode(rw)          SrcSock(rw)

mybrick: ipxDenyTable > DstSockMode=verify DstSock=1111 DstIfStatus=dormant
                        PktTypeMode=dont_verify SrcIfIndexMode=dont_verify

01: ipxDenyDstSockMode.1.1.2( rw):      verify
01: ipxDenyDstSock.1.1.2( rw):          1111
01: ipxDenyDstIfStatus.1.1.2( rw):      dormant
01: ipxDenyPktTypeMode.1.1.2(-rw):      dont_verify
01: ipxDenySrcIfIndexMode.1.1.2( rw):   dont_verify

mybrick : ipxStaticRouteTable> ipxDenyTable

inx PktTypeMode(*-rw)        PktType(rw)          DstIfStatus(rw)
    DstNetMode(rw)           DstNet(rw)           DstNodeMode(rw)
    DstNode(rw)              DstSockMode(rw)      DstSock(rw)
    SrcIfIndexMode(*rw)      SrcIfIndex(rw)       SrcNetMode(rw)
    SrcNet(rw)               SrcNodeMode(rw)      SrcNode(rw)
    SrcSockMode(rw)          SrcSock(rw)

 00 dont_verify              unknown              dormant
    dont_verify              0                    dont_verify
                             verify               1111
    dont_verify              0                    dont_verify
    0                        dont_verify
    dont_verify              0

mybrick:ipxDenyTable>
```

This filter would not allow ISDN connections to be opened for Novell serialization packets. If an ISDN connection is already open, serialization packets would be allowed through. By default this filter is automatically added to the ***ipxDenyTable*** at boot time, and can be removed.

9

Chapter Nine

USING THE BRICK AS A CAPI SERVER

What's Covered?

- Background on CAPI
  - Register Connect Release
  - Message Queues
- The Remote CAPI
  - Remote CAPI Library
  - RVS-COM Lite for Windows 95 and Windows NT
- CAPI Settings on the BRICK
  - CAPI System Tables
  - CAPI TCP Port
- Tracing CAPI Connections
- CAPI Features and Enhancements Supported by the BRICK
  - CAPI 1.1 Enhancements
  - BinTec Extensions to CAPI 1.1
  - CAPI 2.0 Enhancements
  - BinTec Extensions to CAPI 2.0

Background on CAPI

CAPI (Common ISDN Application Programming Interface) is an application programming interface used for developing ISDN applications for various operating systems. These applications utilize a host's connected ISDN interfaces. *RVS-COM (Lite Version) for Windows 95 and Windows NT* which is included with the BRICK is an example of a suite of CAPI applications.

The CAPI specification defines both the CAPI entity (the server) and the protocol that CAPI applications must use when communicating with this entity. By defining a standardized software interface, CAPI allows applications to access ISDN services in a straightforward way. The CAPI specifications are a result of close cooperation among leading ISDN manufacturers and are set forth in two versions.

- CAPI Version 1.1 dated September 1990
- CAPI Version 2.0 dated February 1994

Normally, CAPI is implemented on a single PC with one ISDN adapter and direct access to the ISDN. The CAPI server and the running CAPI applications communicate directly under the shadow of one operating system. Operating systems supporting CAPI include:

- Windows 3.11
- Windows 95
- Windows NT
- Novell
- UNX platforms (using libcapi+capifax)

Register Connect Release

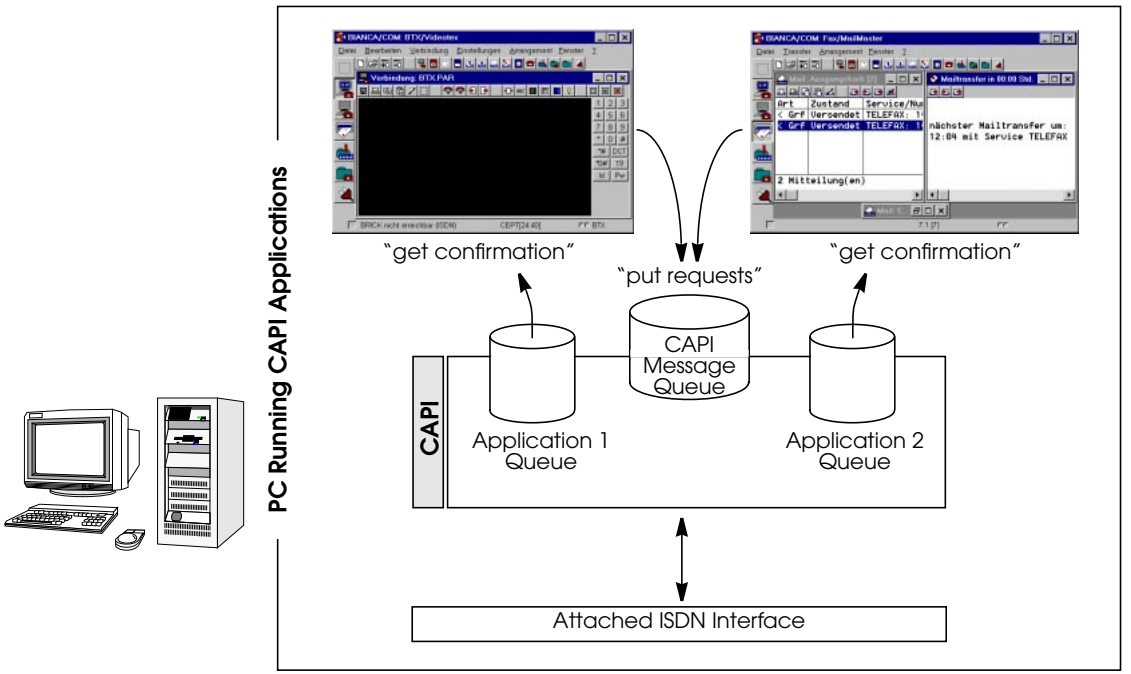
The basic working process of CAPI can be simplified in three basic steps.

1. **Application Registering**
   Before an application can communicate with the CAPI, it must register with the server. Once it's registered, the server assigns it a unique application ID (APPL_ID). At the same time, the application assigns memory space for its message queue (see below).
2. **Application Connections**
   The application is now ready to establish network connections with other applications using the attached ISDN interfaces.
3. **Application Releasing**
   Just before the application is closed it releases its connection with the CAPI server. In other words the application "un-registers" itself with the server.

Message Queues

CAPI applications communicate with the CAPI server via message queues. A message queue can be seen as a sort of one-way pipe. The CAPI server uses a single message queue to accept messages from all CAPI applications. CAPI applications have their own message queues where they receive responses from the CAPI server.

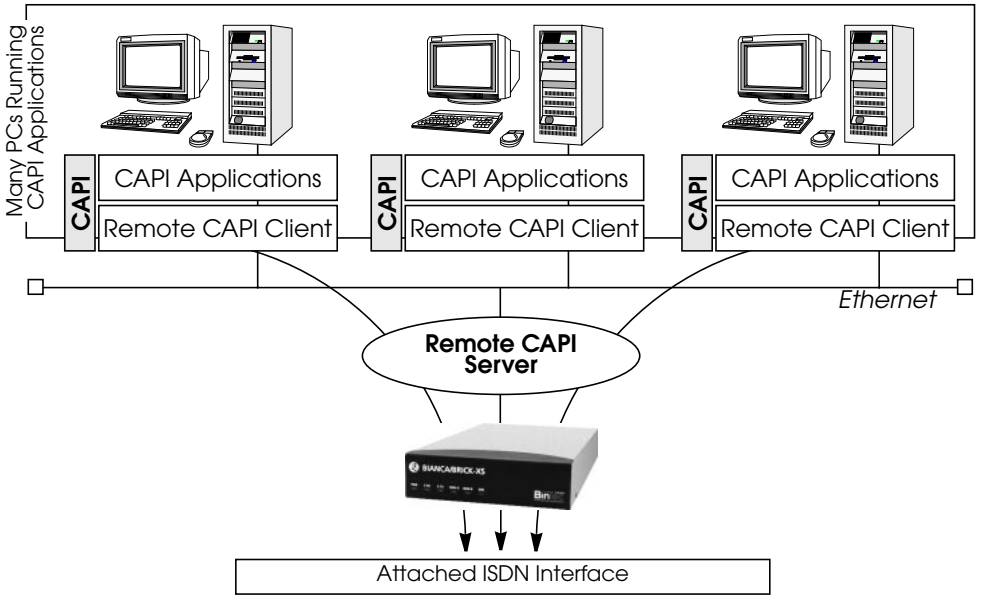
An application issues commands to an ISDN controller by placing a message in the CAPI's message queue and the CAPI returns information to the application via its message queue.



As shown above applications put "requests" in the CAPI server's queue and get "confirmation" messages from the server via their respective queues. Sometimes exchanges are initiated by the server. Here, the server puts indication messages in the application's queue and gets a response message from the application via the CAPI message queue.

The Remote CAPI

Remote CAPI is implemented on the BRICK and extends the CAPI concept to the network level. As mentioned above CAPI and CAPI applications run on a single PC. Remote CAPI is a client-server system that allows distributed applications running on network computers to access the ISDN interfaces of the BRICK. A Remote CAPI client is installed on the PC; the BRICK provides the Remote CAPI Server.



This Remote CAPI is a DualMode server, and supports version 1.1 and 2.0 CAPI. Each PC running Remote CAPI Client software can access the BRICK's ISDN interfaces as if the interfaces were available locally. To CAPI applications, messaging between client and server remains transparent.

Remote CAPI Library

Using the Remote CAPI server and the accompanying Remote CAPI Library for UNIX, or CAPI.DLL for Windows, existing CAPI applications can be ported to the Remote CAPI environment. The remote CAPI library and header file are included on the Companion CD as **libcapi.c** and **libcapi.h.**

RVS-COM Lite for Windows 95 and Windows NT

In addition to the CAPI library, the BRICK comes with the CAPI communications package *RVS-COM (Lite Version) for Windows 95 and Windows NT.* RVS-COM Lite consists of several important data communications programs that have been optimized for the Remote CAPI environment.

CAPI Settings on the BRICK

CAPI System Tables

The CAPI subsystem on the BRICK consists of the following system tables. These tables are intended mainly for keeping track of status information and debugging CAPI connections. Most of these variables are related to CAPI internals and will only be of importance to CAPI developers.

```
mybrick: > g capi

73 capiApplTable     74 capiListenTable     75 capiPlciTable          76 capiNcciTable
77 capiInfoTable     78 capiConfigTable     79 capiMultiControllerTable 80 capiUserTable

mybrick: >
```

capiApplTable	Lists all currently connected CAPI applications.
capiListenTable	Contains application-specific listen settings.
capiPlciTable	Contains additional information about connected applications.
capiNcciTable	Contains information for each CAPI NCCI.
capiInfoTable	Logs the last 10 CAPI Info Codes and their message identifiers. A list of CAPI Info Codes and their values is contained in Appendix A.
CapiConfigTable	Optional settings specific to a local ISDN stack.
capiMultiControllerTable	Contains mappings between controller numbers used by CAPI applications and the ISDN stacks available on the BRICK
capiUserTable	Includes authentication settings that control access to the BRICK's CAPI subsystem.

CapiConfigTable

The ***capiConfigTable*** contains configuration information specific to each ISDN stack on the BRICK. Each table entry defines settings applicable to all CAPI calls[1] that connect over the respective stack (***capiConfigStkNumber***).

The default values for the ***capiConfigTable*** variables are shown below:

```
mybrick: > capiConfigTable

inx StkNumber(*ro)          FaxG3RcvSpeed(rw)   FaxG3ECM(rw)
    FaxG3Header(rw)         VoiceCoding(rw)      SendAlerting(rw)
    V42bis(rw)              ModemDefault(rw)

 00 0                       maximum             off
    logo_header             reverse             voice_only
    off                     modem_profile_1
```

StkNumber As mentioned above, this defines the ISDN stack number the rest of the variables apply to. Stack numbers are numbered from 0 through 31.

FaxG3RcvSpeed The receive speed to use when receiving G3 faxes. If a CM-EBRI is connected to this stack or you are using a V!CAS, be sure to set this field to `maximum`.

FaxG3ECM Specifies whether error correction mode should be used for G3 facsimile transmissions.

FaxG3Header This specifies whether a header-line and/or logo should appear on outgoiing facsimilies. The header--line contains calling information, the logo contains the ***BIANCA/FAX*** symbol.

VoiceCoding Switches the bit order for voice-data.

SendAlerting For CAPI 1.1 this specifies when the CAPI server should send alert messages for incoming calls.

V42bis For V.42bis data compression. Compression is used when ***V42bis***=`on` and the remote side supports V.42bis.

capiConfigModemDefault Specifies the modem profile of the ***mdmProfileTable*** which contains the default modem parameters to use for

1.The only exception is the ***capiConfigSendAlerting*** variable.

modemconnections. The valid range is from **modem_profile_1** to **modem_profile_8**.

capiMultiControllerTable

The ***capiMultiControllerTable*** was added to the CAPI group to enable the use of CAPI with different ISDN controllers at the same time.
This table contains mappings between controller numbers used by CAPI applications and the ISDN stacks available on the BRICK (i.e., the ***Number*** field of the ***isdnStkTable***).
The ***Version*** field specifies whether an entry applies to a capi11 or capi20 application.
If no CAPI 1.1 entry is defined, CAPI 1.1 applications are assigned ***isdnStkNumber*** n where n is the controller number requested by the application.
If no CAPI 2.0 entry is defined, CAPI 2.0 applications are assigned ***isdnStkNumber*** n-1 where n is the controller number requested by the application.

Creating entries: entries are created by assigning a value to the ***capiControllerNumber*** object.

Deleting entries: an entry can be removed by assigning the value ***delete*** to its ***capiControllerVersion*** object.

The fields of the table have the following meanings:

Number	The controller number requested by the CAPI application.
StkMask	This binary number defines the ISDN stack(s) to use for the specified CAPI 1.1 or CAPI 2.0 applications. Each bit corresponds to one entry (stack) in the ***isdnStkTable***, the rightmost bit selects entry 0, the next bit selects entry 1, and so forth. For example, ***Number***=1 ***StkMask***=0b1101 ***Version***=capi11 means: allow CAPI 1.1 applications requesting ISDN controller 1 to use ISDN stacks 0, 2 and 3.
Version	Specifies which CAPI applications (version 1.1, or 2.0) this entry applies to. Set this field to ***delete*** to delete this entry.

capiUserTable

The ***capiUserTable*** and ***isdnDispatchTable*** are both parts of the CAPI User Concept. The CAPI User Concept gives you greater control of access to the BRICK's CAPI subsystem. Each network user that attempts to access the BRICK's CAPI subsystem must first be authenticated by using a user name and password which is also configured on the BRICK, i.e. ***capiUserName*** and ***capiUserPassword*** in the ***capiUserTable*** (it can also be configured over Setup Tool in the ***IP static Settings*** menu). Only if authentication is successful, can the user receive incoming calls or establish outgoing connections via the CAPI.
The fields of the ***capiUsertable*** have the following meanings:

capiUserName	The name of the user. Entered on the BRICK and matches the entry in the Capi Configuration on the BRICKware, as well as the entry in ***isdnDspUserName***.
capiUserPassword	The password of the user. Entered on the BRICK and matches the entry in the Capi Configuration on the BRICKware.
capiUserCapi	Allows or denies the use of CAPI.

When an incoming CAPI call arrives at the BRICK from a WAN partner, the Called Party Number is compared with the ***isdnDspLocalNumber*** in the ***isdnDispatchTable***.

The entire ***isdnDispatchTable*** is only relevant for pure router devices; not for PBX devices.

If the CAPI User Concept is being used, the same user name as configured in ***capiUserName*** is also configured in the ***isdnDspUserName*** in the ***isdnDispatchTable***. The entry in ***isdnDspUserName*** and the entry in ***capiUserName*** are then compared. If they match, the BRICK then forwards the call only to that CAPI application that is authenticated with the same user name as configured in the ***capiUserTable***.

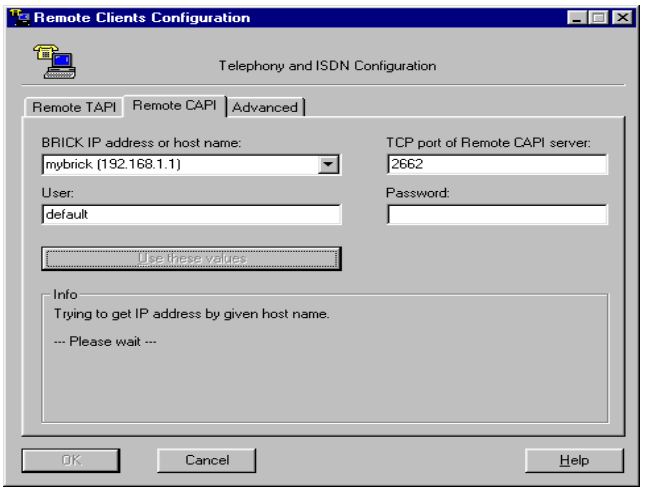
CAPI TCP Port

The only required setting on the BRICK is the CAPI TCP port. This is the port the BRICK listens to for incoming CAPI connections, i.e. it must match the port number used by the Remote CAPI Client software.

The default port is **2662,** however it can be changed by changing the ***biboAdmCapiTcpPort*** variable.

Note that CAPI can be disabled completely on the BRICK by assigning port number 0 to the ***biboAdmCapiTcpPort*** variable.

On the PC the CAPI/TAPI server ports are configured in the program "Remote Clients Configuration". The CAPI Tracer of the DIME Tools can be configured when starting a Trace session (Start/New CAPI Trace).



The current Unix Tools "capitrace", "eft", and "eftd" use CAPI port 6000 as the default setting. The ports of these programs can be changed by setting the environment variable "CAPI_PORT" under Unix. (e.g : CAPI_PORT=2662↵, export CAPI_PORT↵)

Tracing CAPI Connections

When debugging connections on the BRICK you may need to trace the ISDN channels to determine why your CAPI connections may be failling. A tracer collects all CAPI messages exchanged between CAPI applications on the LAN and the BRICK.

Windows 95/Windows NT users will want to use the included *CAPI Trace* program included with *DIME Tools*. *CAPI Tracer* is a CAPI application that communicates directly with the BRICK via a TCP connection; therefore the installation of the Remote CAPI Client is not required on the trace-host (the host where *CAPI Trace* is started from). For information on using *CAPI Trace*, refer to the *BRICKware for Windows* documentation.

For UNIX systems the **`capitrace`** program is also included on the Companion CD. For information on using the capitrace program, refer to Chapter 7, *Command Reference*, in the *User's Guide*.

CAPI Features and Enhancements Supported by the BRICK

The following sections list official enhancements to the CAPI 1.1 and 2.0 standards as well as BinTec specific extensions that are supported by all[1] BRICK products. As noted earlier, this information will mostly be of interest to CAPI developers.

CAPI 1.1 Enhancements

In addition to the official standards for CAPI Version 1.1 remote CAPI on the BRICK supports the following enhancements.

- Use of fax group 3
- Support for X.25 PLP on the D-channel
- CAPI-E-DSS1-Mapping
- Management of semi-permanent connections
- Direct Dial In (DDI) for NT1 equipment
- Extension of CAPI error codes/E-DSS1 adaptation
- Support for DTMF[2] functions (receive only)
- NCPD in accordance with ISO 8208 protocol

BinTec Extensions to CAPI 1.1

- **CAPI-DSS1 Mapping** Which maps the Service Indicator and Additional Info (SI and ADD), according to 1TR6, HLC, and LLC.
- **Specification for V.110 Connections** Inband Negotiation is not implemented for synchronous transmission using bitrate adaptation according to ITU-T V.110 (user-rate: 10101111).
- **DTMF[2]-Capabilitiy** Only the DTMF-detection (DTMF_IND, and DTMF_RESP messages) and B2-Protocol extensions »0x0B« (i.e. Bittranspar-ent-Transmit Only) are supported.
- ***BinTec specific CAPI-Extensions*** Support for 2400 bps modems (V.22bis). The *SELECT_B2_PROTOCOL_REQ* message contains the B2 parameter which

1.With exceptions which are appropriately noted.
2.Dual Tone Multi Frequency is only supported on the V!CAS and products with a CM-1EBRI.

is used by modems.

B2	Protocol
0xf0	analog modem, (BinTec-CAPI only) 2400 bps full-duplex: V.22bis

The CAPI-DLPD[1]-Structure for the identification of the »Modem« protocol is coded (analogous to V.110 DLPD) as follows:

DLPD	Meaning
WORD	DATA-LENGTH
BYTE	LINK-ADDRESS-A (not used)
BYTE	LINK-ADDRESS-B (not used)
BYTE	MODULO (not used)
BYTE	WINDOW-SIZE (not used)
BYTE	User Rate (as with V.110)

The *Userrate* bitfield relevant for modem operation is coded (analogous to userrate in V.110) as follows:

Userrate Byte	Coding
--1- ----	7 Databits
--0- ----	8 Databits
---- 0---	no parity
---- 1---	even parity

1.Data Link Protocol Description

- ***Optional NCPI-Parameters for ISO 8208*** To change the ISO 8208-Default-Windowsize (Standard: 2), these NCPI parameters must be used:

STRUCT NCPI		Default-value
WORD	lic	0
WORD	hic	0
WORD	ltc	1
WORD	htc	1
WORD	loc	0
WORD	hoc	0
BYTE	modulo_mode	8
BYTE	default_window_size (* additional *)	2

- **CAPI 1.1 supported B2/B3 B-channel protocols[1]:**

CAPI 1.1 CAPI_SELECTB2_REQ		BRICK-XM BRICK-XL				BRICK-XS	VICAS
		CM-1BRI	CM-2BRI	CM-1EBRI	CM-PRI		
0x01	X.75 SLP Basis Operation Mode, with Implementation Rules IAW T.90.	•	•	•	•	•	•
0x02	Transparent-HDLC with Bit-Stuffing, Frame-Detection and CRC-Check	•	•	•	•	•	•
0x03	Bit transparent operation	•	•	•	•	•	•
0x05	X.75 Btx	•	•	•	•	•	•
0x06	Fax G3			•			•

CAPI 1.1 CAPI_SELECTB2_REQ		BRICK-XM BRICK-XL				BRICK-XS	V!CAS
		CM-1BRI	CM-2BRI	CM-1EBRI	CM-PRI		
0x07	LAP-D	•	•	•	•	•	•
0x08	V.110 with transparent B2-Protocol			•	•		
0x0a	V.110 with X.75 SLP Basis Operation Mode with Implementation Rules IAW T.90			•	•		
0x0b	Bit transparent operation (transmit only)	•	•	•	•	•	•

CAPI 1.1 CAPI_SELECTB3_REQ		BRICK-XM BRICK-XL				BRICK-XS	V!CAS
		CM-1BRI	CM-2BRI	CM-1EBRI	CM-PRI		
0x01	0x01 T.70 NL for circuit switching (CSPDN) (preset).	•	•	•	•	•	•
0x02	ISO 8208 (DTE/DTE).	•	•	•	•	•	•
0x03	Level 3, IAW T.90, Appendix II.	•	•	•	•	•	•
0x04	Transparent.	•	•	•	•	•	•
0x05	Fax T.30.			•			•

1. DTMF is only supported on the V!CAS and products with a CM-1EBRI.

CAPI 2.0 Enhancements

The BRICK supports the CAPI Version 2.0 standard from February 1994 with the following enhancement.

- Support of Direct Dial In (DDI)

BinTec Extensions to CAPI 2.0

- **CAPI 2.0 supported Layer 1, Layer 2, and Layer 3 B-channel protocols[1]:**

CAPI 2.0 Layer 1 Protocols		BRICK-XS	V!CAS	BRICK-XL				
				BRICK-XM				
				CM-1BRI	CM-2BRI	CM-1EBRI	CM-PRI	FML-8MD
0:	64 kBit/s with HDLC framing. This is the default B1 protocol.	•	•	•	•	•	•	
1:	64 kBit/s bit transparent operation with byte framing from the network.	•	•	•	•	•	•	
2:	V.110 asynchronous operation with start/stop byte framing.					•	•	
3:	V.110 synchronous operation with HDLC framing.					•		
4:	T.30 modem for fax group 3.		•			•		•
6:	56 kBit/s bit transparent operation with byte framing from the network.	•	•	•	•	•	•	

1.DTMF is only supported on the V!CAS and products with a CM-1EBRI.

CAPI 2.0 Layer 2 Protocols		BRICK-XS	VICAS	BRICK-XL				
				BRICK-XM				
				CM-1BRI	CM-2BRI	CM-1EBRI	CM-PRI	FML-8MOD
0:	ISO 7776 (X.75 SLP). This is the default B2 protocol.	●	●	●	●	●	●	
1:	Transparent.	●	●	●	●	●	●	
3:	LAPD IAW Q.921 for D channel X.25	●	●	●	●	●	●	
4:	T.30 for fax group 3.		●			●		●
5:	Point to Point Protocol (PPP).	●	●	●	●	●	●	
7:	Modem with full negotiation		●			●		●

CAPI 2.0 Layer 3 Protocols		BRICK-XS	VICAS	BRICK-XL				
				BRICK-XM				
				CM-1BRI	CM-2BRI	CM-1EBRI	CM-PRI	FML-8MOD
0:	Transparent. This is the default B3 protocol.	●	●	●	●	●	●	
1:	T.90NL with compatibility to T.70NL IAW T.90 Appendix II.	●	●	●	●	●	●	
2:	ISO 8208 (X.25 DTE-DTE).	●	●	●	●	●	●	

CAPI 2.0 Layer 3 Protocols		BRICK-XS	V!CAS	BRICK-XL				
				BRICK-XM				
				CM-1BRI	CM-2BRI	CM-1EBRI	CM-PRI	FML-8MOD
3:	X.25 DCE.	•	•	•	•	•	•	
4:	T.30 for fax group 3.		•			•		•
5:	T.30 for fax group 3 (extended)		•			•		
7:	Modem		•			•		•

10

Chapter Ten

TELEPHONY SERVICES ON THE BRICK

What's Covered?

- Telephony Services on The BRICK
- What is POTS?
  - POTS Interfaces
    - Dispatching Analog Calls
    - Internal Calls
    - External Calls
- What is TAPI?
  - Remote TAPI on the BRICK
    - TAPI Settings
- Configuring Telephony Services
  - Two workspaces: two telephones, one V!CAS
  - One workspace: one V!CAS, one telephone, one fax

Telephony Services on The BRICK

Telephony Service on the BRICK means that you can connect conventioanal analog devices (telephone, fax, modem, etc.) to the BRICK and place or receive analog calls via any of the BRICK's ISDN interfaces.

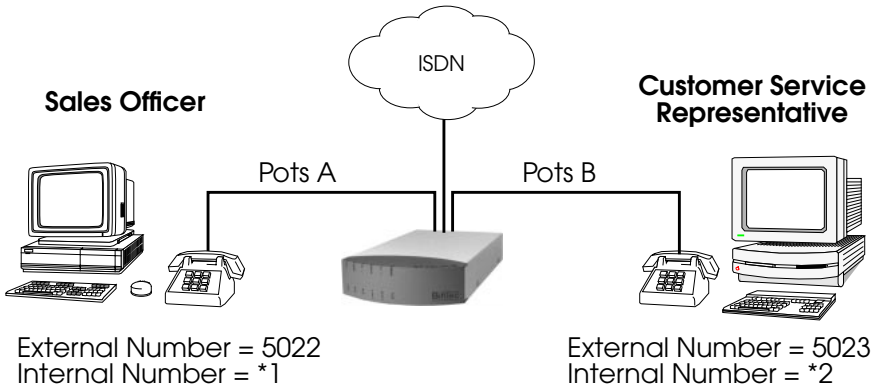
NOTE:

System software support for telephony services is included on all BRICK products. However, to take advantage of telephony services, a CM-AB module must be installed.

This allows you to use the BRICK as a PBX (Private Branch exchange) to:

1. Make toll-free calls internally between connected analog devices, or
2. Place (or receive) calls via the ISDN from connected analog devices.

This could be useful for small offices when combined with PCs running Remote TAPI Client software (included on the Companion CD).



In the simple scenario shown above, the BRICK is used to make inter-office (toll-free) calls between workstations using the internal telephone numbers. In addition, both parties can simultaneously place or receive calls from the ISDN using their respective analog devices.

Although telephony services and the Remote TAPI go together hand in hand, the rest of this chapter describes using the BRICK's POTS ports and using the BRICK as a Remote TAPI Server separately.

What is POTS?

In the networking field the term POTS (Plain Old Telephone Service) is often used to refer to the conventional analog telephone network or analog-based communications devices. With the CM-AB module installed the BRICK has two POTS ports on it's backplane called POTS A and POTS B, for the attachment of such devices (analog telephone, FAX machine, or modem).

POTS Interfaces

When the CM-AB module is installed an entry in the ***biboAdmBoardTable*** will display the slot number where the board is installed. On the V!CAS and the BRICK-XS Office slot 3 is always used. Information about the devices connected to the POTS ports is stored in the ***potsIfTable***. On the V!CAS the table looks as follows.

```
mybrick: > potsIfTable
inx Slot(*ro)          Unit(*ro)          Type(rw)
00  3                  0                  any
01  3                  1                  any
mybrick : potsIfTable>
```

The fields of the ***potsIfTable*** have the following meanings.

Slot	Identifies the slot the CM-AB module is installed in.
Unit	Identifies the port, POTS A = Unit 0, POTS B = Unit 1.
Type	Identifies the types of calls this device will accept. Possible values include: any, telephony, fax, modem, or disable. Disable means that the device can not place or accept calls.

These are the default entries created by the system at boot time upon detection of an installed CM-AB module. ***potsIfTable*** entries can only be removed by the system.

Dispatching Analog Calls

The BRICK dispatches incoming calls (from the ISDN) according to the ISDN Call Dispatching algorithm. The dispatching algorithm distributes calls to BRICK services according to the "Called Party's Address" contained in the call packet and the localNumber field of the ***isdnDispatchTable***. Calls dispatched to the pots service (***Item***=pots) are given to POTS devices based on additional information contained in the service indicator field of the ISDN Call packet and the ***Type*** field mentioned above. The service indicator field simplay specifies the type (FAX, voice, data , etc.) of call.

NOTE

If the call originated from an analog device, the ISDN can't always accurately report the call type and simply reports the call as being an "analog" call; the actual call may be a FAX or voice call.

The different call types and the services they support are as follows:

potsType	Accepts calls from	
	analog network	ISDN devices
any	analog	telephony, fax, modem
telephony	analog	telephony
fax	analog	fax
modem	analog	modem

Internal Calls

Internal calls can be made between devices connected to the BRICK's POTS ports. This requires that each POTS device in the connection is assigned an internal number). Note that these calls are dispatched according to the ***isdnDispatchTable***; therefore it's recommended that you assign internal numbers using the format "***<*internal number*>**" to ensure internal and external MSNs are kept separate. Internal numbers are assigned to devices in the ***isdnDispatchTable*** as follows.

isdnDispatchTable	POTS A	POTS B
StkNumber	**31**	
Item	**pots**	
LocalNumber	*<the telephone number for this device>*	
Bearer	**any**	
Slot	*<slot number for CM-AB module>*	
Unit	**0**	**1**
Direction	**both**	
Mode		
UserName		

NOTE

By default the BRICK automatically creates two dispatch table entries upon detection of the CM-AB module at boot time. The default internal numbers for ports A and B are "*1" and "*2" respectively.

External Calls

External calls can also be placed or received from POTS devices. External numbers are assigned to POTS devices in the same way as for internal numbers with the exception that the ***StkNumber*** field must specify a "real" ISDN Stack number.

Outbound External Calls

Each POTS device must have exactly one outgoing number for outbound external calls. This means either

1. an external number entry with the ***Direction*** field set to **`both`** OR
2. an external number entry with the ***Direction*** set to **`outgoing`**.

Inbound External Calls

A POTS device may be configured to respond to different external numbers by creating multiple external-number entries in the dispatch table and setting the ***Direction*** field to **`incoming`**.

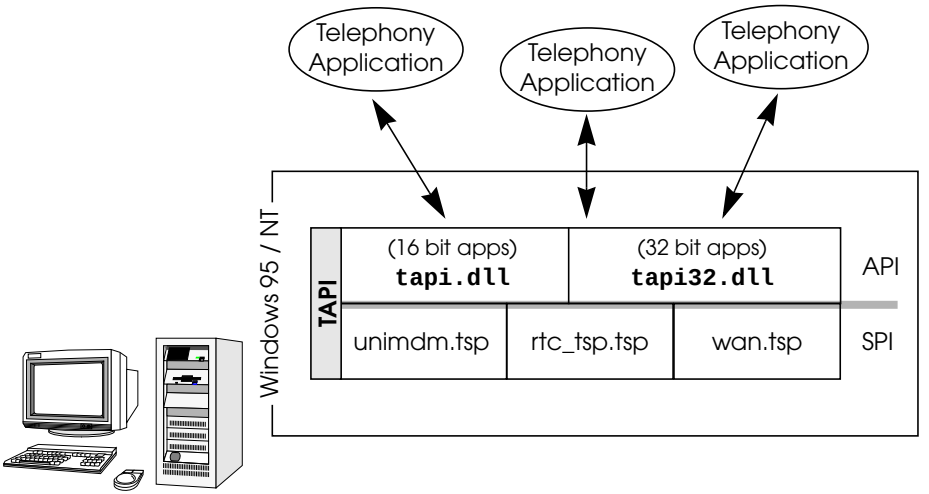
What is TAPI?

TAPI (Telephony Applications Programming Interface) is a programming interface initially defined by Microsoft and Intel for developing Windows-based telephony applications. A telephony application uses attached telephone equipment to place, accept, or monitor calls. The Microsoft Dialer (part of Windows) is an example of a TAPI application.

TAPI actually consists of two parts.

1. The API defines how applications (like the Microsoft Dialer) interact with the underlying operating system (Windows 95 or NT).
   It gives applications access to Windows' telephony features.
2. The SPI (Service Provider Interface) defines how the operating system interacts with attached telephony hardware. More than one TSP (Telephone Service Provider) may be installed on the PC, each one specifies how the OS communicates with a paticular piece of hardware.

TAPI on Windows 95 and NT systems looks like this.



Both dlls are shipped with Win 95 and NT. Windows uses the appropriate dll depending on the user's application (16 or 32 bit).

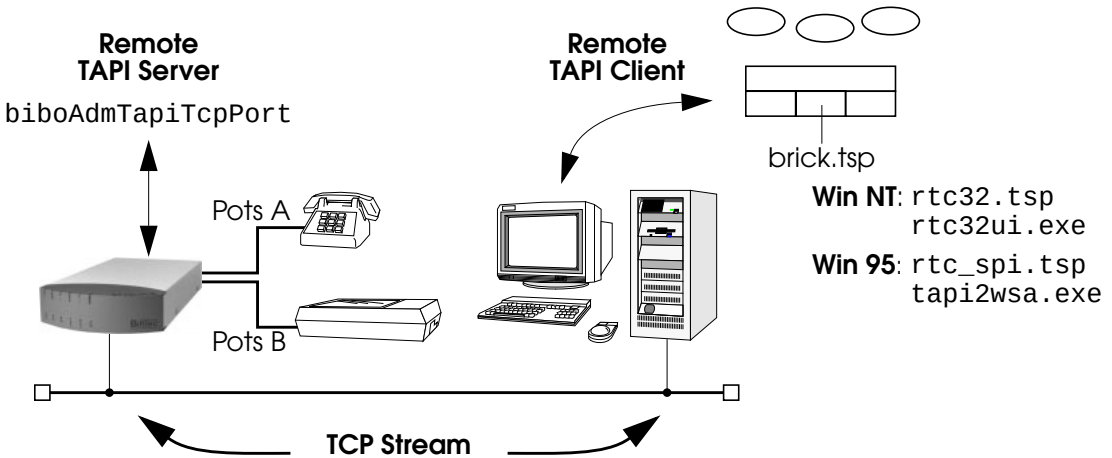
Hardware specific `*.tsp` and `*.exe` files provided by the equipment manufacturer.

Remote TAPI on the BRICK

The BRICK can be used as a remote TAPI server, meaning that it can place, accept, and route calls from a PC on the LAN where the Remote TAPI Client is installed. Remote TAPI Client software for PCs is included on the Companion CD and is installed from the BRICKware installation program.

The Remote TAPI Client forwards all TAPI requests made by TAPI applications to the BRICK via a TCP stream.

The BRICK accepts TAPI client connections via it's TAPI port.



TAPI Settings

Configuring the BRICK as a TAPI server is straightforward. On the BRICK, all that is required is that the BRICK's TAPI port be set. This is defined in the ***admin*** table.

The ***biboAdmTapiTcpPort*** variable defines the TCP port on the BRICK remote TAPI applications must connect to. By default the BRICK uses TCP Port 6001. The same value must be configured on the PC running the Remote TAPI Client program.

The TAPI server can be disabled completely by setting the TAPI port to 0.

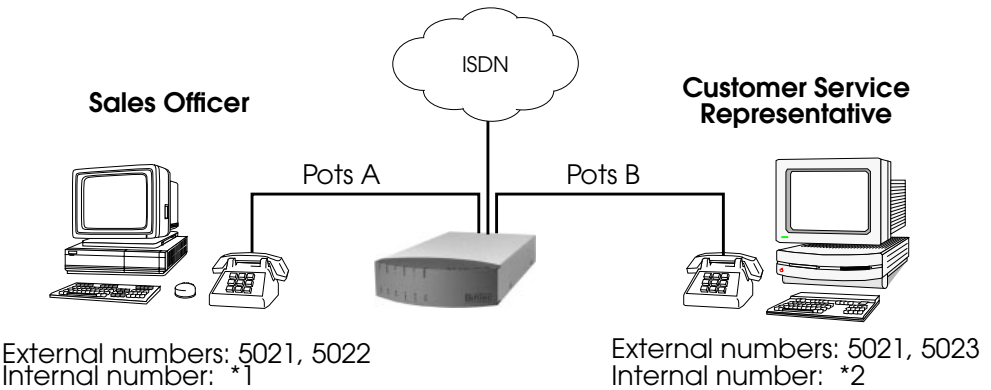
Configuring Telephony Services

Below are two example configurations showing how the BRICK can be used as a simple PBX. The Remote TAPI Client could be installed on the computers in these examples allowing calls to be managed directly from the PC.

Two workspaces: two telephones, one V!CAS

Here we have two workspaces, a Sales Agent and a Customer Service Representative. Each workspace has an analog telephone that is connected to the BRICK. The sample configuration shows the entries that would be made to the ***isdnDispatchTable*** to allow:

1. Both workspaces receive incoming ISDN calls placed to 5021,
   i.e., both phones ring, the first one to pick up gets the call.
2. Each workspace has a separate number for direct calls.
3. Internal calls can be placed between workspaces.



Step 1

Create the internal numbers by locating the entries for the POTS A and POTS B interfaces in the ***isdnDispatchTable***. There will be two entries that use ***Stack*** 31. You can identify the A and B interfaces by the ***Slot***:***Unit*** combination. POTS A is always at Unit 0, and POTS B at Unit 1. By default, POTS A uses the internal number "*1" while POTS

B uses "*2". Since the default values are what we want for our our example setup, we don't need to change them.

```
mybrick: > isdnDispatchTable
inx StkNumber(*rw)          Item(*-rw)          LocalNumber(rw)
    LocalSubaddress(rw)     Bearer(rw)          Slot(rw)
    Unit(rw)                Direction(rw)       Mode(rw)
    UserName(rw)

00 31                       pots                "*1"
                            any                 3
    0                       both                right_to_left
    "default"

01 31                       pots                "*2"
                            any                 3
    1                       both                right_to_left
    "default"
mybrick : isdnDispatchTable>
```

Step 2 Create an entry for each device's direct number. Only POTS A can answer calls to 5022; only POTS B can answer calls to "5023". The first entry is for POTS A, the second POTS B.

```
mybrick: isdnDispatchTable > StkNumber=0 Item=pots Slot=3 Unit=0 LocalNumber=5022
                             Direction=both Mode=right_to_left UserName="default

mybrick: isdnDispatchTable > StkNumber=0 Item=pots Slot=3 Unit=1 LocalNumber=5023
                             Direction=both Mode=right_to_left UserName="default


mybrick : isdnDispatchTable> isdnDispatchTable
inx StkNumber(*rw)           Item(*-rw)          LocalNumber(rw)
    LocalSubaddress(rw)      Bearer(rw)          Slot(rw)
    Unit(rw)                 Direction(rw)       Mode(rw)
    UserName(rw)

02 0                         pots                "5022"
                             any                 3
    0                        both            right_to_left
    "default"

03 0                         pots                "5023"
                             any                 3
    1                        both            right_to_left
    "default"


mybrick : isdnDispatchTable>
```

Step 3

Create the entries for our common external number "5021." This will allow both device to receive calls placed to this number. Again, our first entry is for POTS A the second is for POTS B.

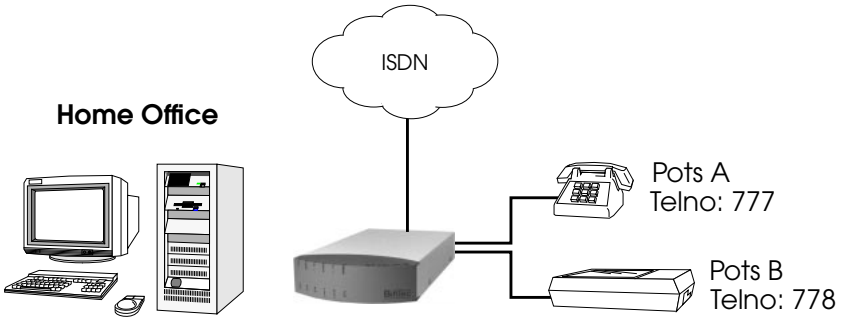
```
mybrick: isdnDispatchTable > StkNumber=0 Item=pots Slot=3 Unit=0 LocalNumber=5021
                                    Direction=incoming Mode=right_to_left User-
Name="default

mybrick: isdnDispatchTable > StkNumber=0 Item=pots Slot=3 Unit=1 LocalNumber=5021
                                    Direction=incoming Mode=right_to_left User-
Name="default

mybrick : isdnDispatchTable> isdnDispatchTable
inx StkNumber(*rw)                  Item(*-rw)          LocalNumber(rw)
    LocalSubaddress(rw)             Bearer(rw)          Slot(rw)
    Unit(rw)                        Direction(rw)       Mode(rw)
    UserName(rw)

04 0                                pots                "5021"
                                    any                 3
    0                               incoming        right_to_left
    "default"

05 0                                pots                "5021"
                                    any                 3
    1                               incoming

mybrick : isdnDispatchTable>
```

The configuration is complete. Don't forget to save your changes to a configuration file with **`cmd=save`**.

One workspace: one V!CAS, one telephone, one fax

In this example, we have a workspace consisting of a V!CAS, an analog telephone, and a FAX device connected to the V!CAS. In this scenario we want to configure the BRICK so that:

1. Incoming telephony calls are given to the device at POTS A.
2. Incoming FAX calls are given to the device at POTS B.



Step 1

For this example all we need to do is create the external numbers so that both devices can be reached from the ISDN. Although we won't be making Internal Calls in this example, the "Stack 31" entries will still be present in the ***isdnDispatchTable***.

```
mybrick: > isdnDispatchTable
inx StkNumber(*rw)          Item(*-rw)          LocalNumber(rw)
    LocalSubaddress(rw)     Bearer(rw)          Slot(rw)
    Unit(rw)                Direction(rw)

02 0                        pots                "777"
                            telephony           3
    0                       both

03 0                        pots                "778"
                            fax                 3
    1                       both

mybrick : isdnDispatchTable>
```

A

Appendix A

CAPI INFORMATION VALUES

CAPI 1.1 and CAPI 2.0 info values with their appropriate error-codes associated with the ***capiInfoNumber*** field are as follows. Both CAPI 1.1 and 2.0 info values are arranged by error class.

What's Covered?

CAPI 1.1 Info Values

Error Class 10

Formal error messages and errors during management of Message-Queues.

0x1001	Application registration error
0x1002	Wrong application id
0x1003	Message error
0x1004	Wrong capi command
0x1005	Message queue full
0x1006	Message queue empty
0x1007	Messages lost
0x1008	Error during deinstallation

Error Class 20

Addressing errors.

0x2001	Wrong controller
0x2002	Wrong PLCI
0x2003	Wrong NCCI
0x2004	Wrong type

Error Class 31

Incorrect parameters.

0x3101	B-channel incorrectly coded
0x3102	Info mask incorrectly coded
0x3103	Service SI mask incorrectly coded
0x3104	Service EAZ mask incorrectly coded},
0x3105	B2 protocol incorrect

0x3106	Dlpd incorrect
0x3107	B3 protocol incorrect
0x3108	NCPD incorrect
0x3109	NCPI incorrect
0x310a	Flags incorrectly coded

Error Class 32

Unsupported parameters.

0x3201	Controller error
0x3202	Conflict between registrations
0x3203	Function is not supported
0x3204	PLCI not active
0x3205	NCCI not active
0x3206	B2 protocol not supported
0x3207	Change of B2 protocol not possible in this state
0x3208	B3 protocol not supported
0x3209	Change of B3 protocol not possible in this state
0x320a	Parameters used not supported in DLPD
0x320b	Parameters used not supported in NCPD
0x320c	Parameters used not supported in NCPI
0x320d	Data length not supported
0x320e	DTMF number unknown

Error Class 33

Network errors.

0x3301	Error on setup of D-channel layer 1

0x3302	Error on setup of D-channel layer 2
0x3303	Error on setup of B-channel layer 1
0x3304	Error on setup of B-channel layer 2
0x3305	Abort D-channel layer 1
0x3306	Abort D-channel layer 2
0x3307	Abort D-channel layer 3
0x3308	Abort B-channel layer 1
0x3309	Abort B-channel layer 2
0x330a	Abort B-channel layer 3
0x330b	Re-establish B-channel layer 2
0x330c	Re-establish B-channel layer 3

Error Class 34

Network messages (whereby xx refers to the related 1TR6 Error-Cause).

0x3400	Normal call clearing
0x3480	Normal call clearing
0x3481	Invalid call reference value
0x3483	Bearer service not implemented
0x3487	Call identity does not exist
0x3488	Call identity in use
0x348a	No channel available
0x3490	Requested facility not implemented},
0x3491	requested facility not subscribed
0x34a0	Outgoing calls barred
0x34a1	User Busy
0x34a2	negative GBG Comparison

0x34a5	as SPV not aloud
0x34b0	Reverse charging not allowed at origination
0x34b1	Reverse charging not allowed at destination
0x34b2	Reverse charging rejected
0x34b5	Destination not obtainable
0x34b8	Number changed
0x34b9	Out of order
0x34ba	User not responding
0x34bb	User access busy
0x34bd	Incoming calls barred
0x34be	Call rejected
0x34d9	Network congestion
0x34da	Remote user initiated
0x34f0	Local procedure error
0x34f1	Remote procedure error
0x34f2	Remote user suspended
0x34f3	Remote user resumed
0x34ff	User info discarded locally

Error Class 40

FAX-G3 Errors.

0x4001	Remote station is not a fax G3 machine
0x4002	Local fax module busy
0x4003	Disconnected during transfer (remote abort)
0x4004	Disconnected before transfer (training error)
0x4005	Disconnected during transfer (local tx data underrun)

0x4006	Fax module temporary disabled
0x4007	Local disconnect (local abort)
0x4008	Disconnect during transfer (remote procedure error)
0x4009	Remote disconnect (remote abort)
0x400a	Line Disconnect during transfer
0x400b	Disconnect before transfer
0x400c	Local Disconnect (SFF coding error)

CAPI 2.0 Info Values

Error Class 00

Informative values (corresponding message was processed).

0x0001	NCPI not supported by current protocol, NCPI ignored
0x0002	Flags not supported by current protocol, flags ignored
0x0003	Alert already sent by another application

Error Class 10

Error information concerning CAPI_REGISTER.

0x1001	Too many applications
0x1002	Logical block size too small, must be at least 128 bytes
0x1003	Buffer exceeds 64 kByte
0x1004	Message buffer size too small, must be at least 1024 byte
0x1005	Max. number of logical connections not supported
0x1006	Reserved
0x1007	The message could not be accepted because of an internal busy condition
0x1008	OS Resource error (e.g. no memory)

0x1009	COMMON-ISDN-API not installed
0x100a	Controller does not support external equipment
0x100b	Controller does only support external equipment

Error Class 11

Error information concerning message exchange functions.

0x1101	Illegal application number
0x1102	Illegal command or subcommand or message length less than 12 octets
0x1103	The message could not be accepted because of a queue full condition
0x1104	Queue is empty
0x1105	Queue overflow, a message was lost
0x1106	Unknown notification parameter
0x1107	The message could not be accepted because of an internal busy condition
0x1108	OS resource error (e.g. no memory)
0x1109	COMMON-ISDN-API not installed
0x110a	Controller does not support external equipment
0x110b	Controller does only support external equipment

Error Class 20

Error information concerning resource/coding problems.

0x2001	Message not supported in current state
0x2002	Illegal Controller/PLCI/NCCI
0x2003	Out of PLCI
0x2004	Out of NCCI

0x2005	Out of LISTEN
0x2006	Out of FAX resources (protocol T.30)
0x2007	Illegal Message parameter coding

Error Class 30

Error information concerning requested services.

0x3001	B1 protocol not supported
0x3002	B2 protocol not supported
0x3003	B3 protocol not supported
0x3004	B1 protocol parameter not supported
0x3005	B2 protocol parameter not supported
0x3006	B3 protocol parameter not supported
0x3007	B protocol combination not supported
0x3008	NCPI not supported
0x3009	CIP Value unknown
0x300a	Flags not supported (reserved bits)
0x300b	Facility not supported
0x300c	Data length not supported by correct protocol
0x300d	Reset procedure not supported by current protocol

Error Class 33

Protocol error reasons.

0x3301	Protocol error layer 1 (broken line or B-channel removed by signalling protocol)
0x3302	Protocol error layer 2
0x3303	Protocol error layer 3
0x3304	Another application got that call

0x3311	Connecting not successful (remote station is no FAX G3 machine)
0x3312	Connecting not successful (training error)
0x3313	Disconnected before transfer (remote station does not support transfer mode, e.g. resolution)
0x3314	Disconnected during transfer (remote abort)
0x3315	Disconnected during transfer (remote procedure error, e.g. unsuccessful repetition of T.30 commands)
0x3316	Disconnected during transfer (local tx data underrun)
0x3317	Disconnected during transfer (local rx data overflow)
0x3318	Disconnected during transfer (local abort)
0x3319	Illegal parameter coding (e.g. SFF coding error)

Error Class 34

Disconnect cause from the network according to ETS 300102-1/Q.931. In the field 'xx' the cause value received within a cause information element (octet 4) from the network is indicated.

0x3481	Unallocated (unassigned) number
0x3482	No route to specified transit network
0x3483	No route to destination
0x3486	Channel unacceptable
0x3487	Call awarded and being delivered in an established channel
0x3490	Normal call clearing
0x3491	User busy
0x3492	No user responding
0x3493	No answer from user (user alerted)
0x3495	Call rejected
0x3496	Number changed
0x349a	Non-selected user clearing

0x349b	Destination out of order
0x349c	Invalid number format
0x349d	Facility rejected
0x349e	Response to STATUS ENQUIRY
0x349f	Normal, unspecified
0x34a2	No circuit / channel available
0x34a6	Network out of order
0x34a9	Temporary failure
0x34aa	Switching equipment congestion
0x34ab	Access information discarded
0x34ac	Requested circuit / channel not available
0x34af	Resources unavailable, unspecified
0x34b1	Quality of service unavailable
0x34b2	Requested facility not subscribed
0x34b9	Bearer capability not authorized
0x34ba	Bearer capability not presently available
0x34bf	Service or option not available, unspecified
0x34c1	Bearer capability not implemented
0x34c2	Channel type not implemented
0x34c5	Requested facility not implemented
0x34c6	Only restricted digital information bearer capability is available
0x34cf	Service or option not implemented, unspecified
0x34d1	Invalid call reference value
0x34d2	Identified channel does not exist
0x34d3	A suspended call exists, but this call identity does not
0x34d4	Call identity in use
0x34d5	No call suspended

0x34d6	Call having the requested call identity has been cleared
0x34d8	Incompatible destination
0x34db	Invalid transit network selection
0x34df	Invalid message, unspecified
0x34e0	Mandatory information element is missing
0x34e1	Message type non-existent or not implemented
0x34e2	Message not compatible with call state or message type non-existent or not implemented
0x34e3	Information element non-existent or not implemented
0x34e4	Invalid information element contents
0x34e5	Message not compatible with call state
0x34e6	Recovery on timer expiry
0x34ef	Protocol error, unspecified
0x34ff	Interworking, unspecified

B

Appendix B

ETHERNET FRAMING

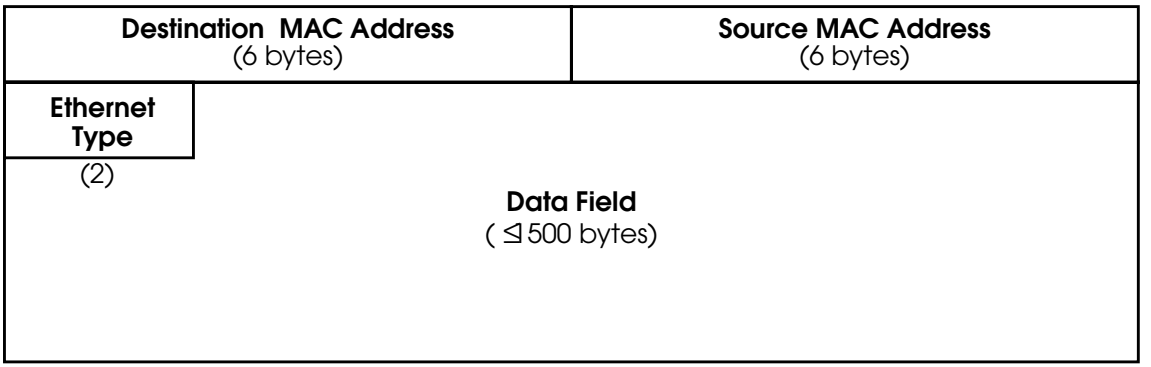
What's Covered?

- Ethernet Framing Types
  - Ethernet II
  - Ethernet LLC
  - Novell 802.3
  - Ethernet SNAP
  - Token Ring

Ethernet Framing Types

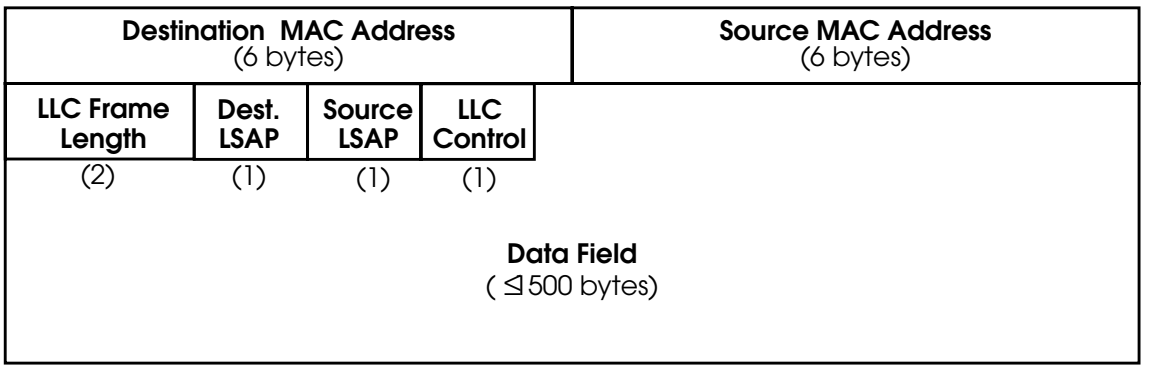
Ethernet II

The **en1** interface can be used for IP and IPX traffic. When using this interface, the following header information is added to the beginning of each data packet.



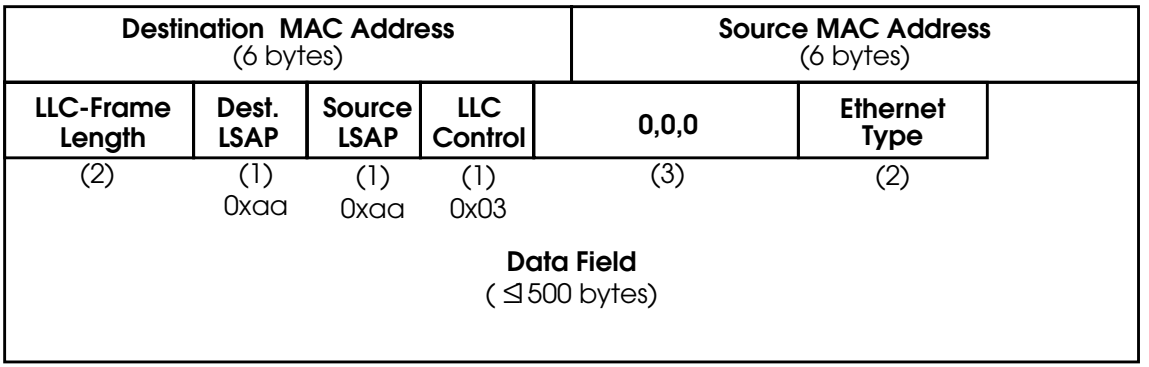
Ethernet LLC

The **en1-llc** interface can be used for X.25, IPX, and Bridging traffic. The following header is added to frames sent over this interface.



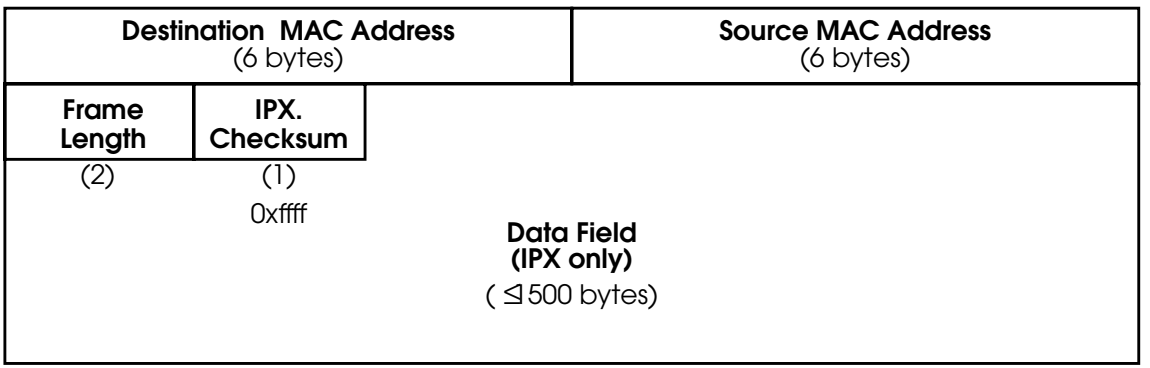
Ethernet SNAP

The **en1-snap** interface can be used for IP and IPX traffic. When using this interface, the following header is added to all frames.



Novell 802.3

The **en1-nov802.3** interface is intended specifically for the IPX protocol. The following header is added to the beginning of IPX frames.



Token Ring

The **en1-tr** interface is intended specifically for token ring

Destination MAC Address (6 bytes)				Source MAC Address (6 bytes)	
LLC-Frame Length	**Dest. LSAP**	**Source LSAP**	**LLC Control**	**0,0,0**	**Ethernet Type**
(2)	(1) 0xaa	(1) 0xaa	(1) 0x03	(3)	(2)
Data Field (≤8192 bytes)					

C

Appendix C

ISDN ERROR CODES

ISDN errors are reported in the ***isdnCallHistoryTable***. Errors originating from the ISDN are reported in the *DSS1Cause* and *1TR6Causefc* fields, depending on which service you're using (Euro ISDN and National ISDN repectively). Errors originating locally on the BRICK are reported in the *LocalCause* field.

What's Covered?

- Local Causes (BRICK)
- DSS1 Causes (Euro ISDN)
  - Resource unavailable class
  - Service/option not available class
  - Service/option not implemented class
  - Invalid message class
  - Protocol error class
  - Internetworking class
- 1TR6 Causes (National ISDN)

Local Causes (BRICK)

Local causes are reported in the *LocalCause* field of the ***isdnCallHistoryTable***

0x01:	ipi: Unknown primitive
0x02:	ipi: Outstate Message has been sent at inappropriate state of call reference.
0x03:	ipi: Mandatory information element (IE) missing
0x04:	ipi: IE not allowed
0x05:	ipi: IL_LOOK
0x06:	ipi: No link or L2 error ISDN-cable is not connected, or ISDN-connection is not available or Layer 2 Connection can not be established See isdniftable for Layer 1 details (Layer1State=F7 means connected i.e. Layer 1 is available). You may also use the program bricktrace to display Layer 2 protocol
0x07:	ipi: All call references are used
0x08:	ipi: Provider has not enough memory
0x09:	ipi: Provider is not ready
0x0a:	ipi: Busy An attempt was made to switch off a busy provider.
0x0b:	ipi: Channel busy
0x0c:	ipi: L3 timer for incoming calls: The call is received but nobody is responding to this call (please verify if the destination address and the LocalNumber of isdndispatchtable are corresponding)
0x0d:	ipi: L3 restart
0x0e:	ipi: L3 error
0x0f:	ipi: L1 error
0x10:	no controller available
0x11:	another application got the call
-1:	no information available

DSS1 Causes (Euro ISDN)

DSS1 causes are reported in the *DSS1Cause* field of the ***isdnCallHistoryTable***

0x80:	Normal call clearing No error occurred.
0x81:	Unallocated (unassigned) number This cause indicates that the destination, requested by the calling user cannot be reached because, although the number is in a valid format, it is not currently assigned (allocated).
0x82:	No route to specified transit network This cause indicates that the equipment sending this cause has received a request to route the call through a particular transit network which it does not recognise. The equipment sending this cause does not recognise the transit network either, because the transit network does not exist or because that particular, while it does exist, does not service the equipment which is sending this cause. This cause is supported on a network-dependent basis.
0x83:	No route to destination This cause indicates that the called user can not be reached because the network through which the call has been routed does not serve the destination desired. This cause is supported on a network-dependent basis.
0x86:	Channel unacceptable This cause indicates the channel most recently identified is not acceptable to the sending entity for use in this call.
0x87:	Call awarded & being delivered This cause indicates that the user has been awarded the incoming call, and that the incoming call is being connected to a channel already established to that similar calls (e.g. packet-mode X.25 virtual calls).
0x90:	Normal call clearing This cause indicates that the call is being cleared because one of the users involved in the call has requested that the call be cleared.
0x91:	User busy This cause is used when the called user has indicated the inability to accept another call.

0x92:	User not responding This cause is used when a user does not respond to a call establishment message with either an alerting or connect indication within the prescribed period of time allocated.
0x93:	No answer from user (user alerted) This cause is used when a user has provided an alerting indication but has not provided a connect indication within a prescribed period o f time.
0x95:	Call rejected This cause indicates that the equipment sending this cause does not wish to accept this call, although it could have accepted the call because the equipment sending this cause is neither busy nor incompatible.
0x96:	Number changed This cause is returned to a calling user when the called party number indicated by the calling user is no longer assigned.
0x9a:	Non-selected user clearing This cause indicates that the user has not been awarded the incoming call.
0x9b:	Destination out of order This cause indicates that the destination indicated by the user can not be reached because the interface to the destination is not functioning correctly. The term not functioning correctly indicates that signalling message was unable to be delivered to the remote user; e.g. a physical layer or data link layer failure at the remote user, user-equipment off-line.
0x9c:	Invalid number format This cause indicates that the called user can not be reached because the called party number is not in a valid format or is not complete.
0x9d:	Facility rejected This cause is returned when a facility requested can not be provided by the network.
0x9e:	Response to STATUS ENQUIRY This cause included in the STATUS message when the reason for generating the STATUS message was the prior receipt of a STATUS ENQUIRY message.
0x9f:	Normal, unspecified This caused is used to report a normal event only when no other cause in the normal class applies.

Resource unavailable class

0xa2:	No circuit/channel available This cause indicates that there is no appropriate circuit/channel presently available to handle the call.
0xa6:	Network out of order This cause indicates that the network is not functioning correctly and that the condition is likely to last a relatively long period of time; e.g. immediately reattempting the call is not likely to be successful.
0xa9:	Temporarily failure This cause indicates that the network is not functioning correctly and that the condition is not likely to last a long period of time; e.g. the user may wish to try another call attempt almost immediately.
0xaa:	Switching equipment congestion This cause indicates that the switching equipment generating this cause is experiencing a period of high traffic.
0xab:	Access Information discarded This cause indicates that the network could not deliver access information to the remote user as requested. i.e. a user-to-user information, low layer compatibility, high layer compatibility or subaddress as indicated in the diagnostic.
0xac:	Requested circuit/channel not available This cause is returned when the circuit or channel indicated by the requesting entity cannot be provided by the other side of the interface.
0xaf:	Resources unavailable, unspecified This cause is used to report a resource unavailable event only when no other cause in the resource unavailable class applies.

Service/option not available class

0xb1:	Quality of service unavailable This cause is used to report that the requested quality of service, as defined in the ITU-T recommendation X.213, cannot be provided (e.g. throughput or transit delay cannot be supported).
0xb2:	Requested facility not subscribed This cause indicates that the requested supplementary service could not be provided by the network because the user has not completed the necessary administrative arrangements with its supporting network.
0xb9:	Bearer Capability not authorized This cause indicates that the user has requested a bearer capability which is implemented by the equipment which generated this cause but the user is not authorized to use.
0xba:	Bearer Capability not presently available This cause indicates that the user has requested a bearer capability which is implemented by the equipment which generated this cause but which is not available this time.
0xbf:	Service or option not available, unspec. This cause is used to report a service or option not available event only when no other cause in the service or option not available class applies.

Service/option not implemented class

0xc1:	Bearer capability not implemented This cause indicates that the equipment sending this cause does not support the bearer capability requested.
0xc2:	Channel type not implemented This cause indicates that the equipment sending this cause does not support the channel type requested.
0xc5:	Requested facility not implemented This cause indicates that the equipment sending this cause does not support the requested supplementary service.

0xc6:	Only restricted digital info. bearer cap. is available This cause indicates that one equipment has requested an unrestricted bearer services but that the equipment sending this cause only supports the restricted version or the restated bearer capability.
0xcf:	Service or option not implemented, unspecified This cause is used to report a service or option not implemented event only when no other cause in the service or option not implemented class applies.

Invalid message class

0xd1:	Invalid call reference value This cause indicates that the equipment sending this cause has received a message with a call reference which is not currently in use on the user-network interface.
0xd2:	Identified channel does not exist This cause indicates that the equipment sending this cause has received a request to use a channel not activated on the interface for a call. For example, if a user has subscribed to those channels on a primary rate interface numbered from 1 to 12 and the user equipment or the network attempts to use channels 13 to 30 this cause is generated.
0xd3:	A suspended call exist, but call identity does not This cause indicates that a call resume has been attempted with a call identity which differs from that in use for any presently suspended call(s).
0xd4:	Call identity in use This cause indicates that the network has received a call suspend request. The call suspend request contained a call identity (including the null call identity) which is already in use for a suspended call within the domain of interfaces over which the call might be resumed.
0xd5:	No call suspended This cause indicates that the network has received a call resume request. The call resume request contained a call identity information element which presently does not indicate any suspended call within the domain of interfaces over which calls may be resumed.
0xd6:	Call with the requested call id has been cleared This cause indicates that the network has received a call resume request. The call

resume request contained a call identity information element which once indicated a suspended call; however, that suspended call was cleared while suspended (either by network timeout or by the remote user).

0xd7:	Service or option not available This cause indicates that the requested service or option is not available or is rejected.
0xd8:	Incompatible destination This cause indicates that the equipment sending this cause has received a request to establish a call which has a low layer compatibility, high layer compatibility or other compatibility attributes (e.g. data rate) which cannot be accommodated.
0xdb:	Invalid transit network selection This cause that a transit network identification was received, which is of an incorrect format.
0xdf:	invalid message, unspecified This cause is used to report an invalid message event only when no other cause in the invalid message class applies.

Protocol error class

0xe0:	Mandatory information element is missing This cause indicates that the equipment sending this cause has received a message which is missing an information element which must be present in the message before that message can be processed.
0xe1:	message type non-existent or not implemented This cause indicates that the equipment sending this cause has received a message with a message type it does not recognise either because this is a message not defined or defined but not implemented by the equipment sending this cause.
0xe2:	message not compatible with call state This cause indicates that the equipment sending this cause has received a message such that the procedures do not indicate that this is a permissable message to receive while in the call state, or a STATUS message was received indicating an incompatible call state.
0xe3:	Information element non-existent or not implemented This cause indicates that the equipment sending this cause has received a message which includes information elements not recognised because the information ele-

	ment identifier is not defined or it is defined but not implemented by the equipment sending the cause. However, the information element is not required to be present in the message in order for the equipment sending the cause to process the message.
0xe4:	invalid information element contents This cause indicates that the equipment sending this cause has received an information element which it has implemented; however, one or more fields in the information element are coded in such a way which has not been implemented by the equipment sending this cause.
0xe5:	message not compatible with call state This cause indicates that the equipment sending this cause has been received which is incompatible with the call state.
0xe6:	recovery on timer expiry This cause indicates that a procedure has been initiated by the expiry of a time in association with error handling procedures.
0xef:	Protocol error, unspecified This cause is used to report a protocol error event only when no other cause in the protocol error class applies.

Internetworking class

0xff:	Internetworking, unspecified This cause indicates that there has been interworking with a network which does not provide causes for functions it takes; thus the precise cause for a message which is being sent cannot be ascertained

1TR6 Causes (National ISDN)

1TR6 causes are reported in the *1TR6Cause* field of the ***isdnCallHistoryTable***

0x81:	Invalid call reference value
0x83:	Bearer service not implemented indicates that a connection with a specific Service Indicator (SI) cannot be set up, either at the calling or called party's end Certain SIs have to be applied for and cleared for ISDN basic access and for PBXs
0x87:	Call identity does not exist The call identity used does not bear relation to any call available for matching. This happens for instance when an attempt is made to resume a call using the function IL_RESUME with the wrong call identity
0x88:	Call identity in use
0x8a:	No Channel available All available B channels of the ISDN access are occupied (either with other end-devices on the S0 bus or other connections on the same end-device.
0x90:	Requested facility not implemented
0xa0:	Outgoing calls barred Outgoing calls are barred from this ISDN access. This is often the case with PBXs if they are restricted locally or nationally.
0xa1:	User Busy Corresponds to an engaged tone on a telephone. All the called party's B channels are busy, the exchange is overloaded or no free trunk was available from the local PBX. (Used instead of 0xd9).
0xa2:	CUG, access denied The called party is member of a closed user group. In these user groups, which are set up by the German Post Office in Germany, a call is only connected if the caller is member of the same group.
0xa3:	Non existent CUG
0xa5:	not permitted as SPV
0xb0:	Reverse charging not allowed at origination
0xb1:	Reverse charging not allowed at destination
0xb2:	Reverse charging rejected

0xb5:	Destination not obtainable The same meaning as "The number you have dialled is not available, please try again" message. Also appears (after timeout) when the telephone number was incomplete.
0xb8:	Number changed Occurs when the called number was too long. This can be the case when the EAZ was included in the dialled number.
0xb9:	Out of order The number dialled is out of order or temporarily not available.
0xba:	User not responding The called party is not accepting or responding to the call for one of the following reasons: - The subscriber's end-device is not switched on or not connected - The required EAZ was not communicated - An incorrect number was dialled - The required SI was not communicated
0xbb:	User access busy Corresponds to an engaged tone on a telephone. All the called party's B channels are busy, the exchange is overloaded or the PBX could not get a free trunk line (instead of 0xd9).
0xbd:	Incoming calls barred Incoming calls are not allowed for the called party (barred) or are not possible (e.g. because the line is out of order).
0xbe:	Call rejected The called party has refused the call or is not allowed to receive the call.
0xd9:	Network congestion The ISDN network is congested (all-trunks-busy condition) at some point. This can occur when calling a PBX and all trunk lines are engaged
0xda:	Remote user initiated
0xf1:	Remote procedure error
0xf2:	Remote user suspended
0xf3:	Remote user resumed
0xf4:	User info discarded locally

D

Appendix D

SYSLOG MESSAGES

What's Covered?

- System Messages
  - ISDN
  - IPX
  - CAPI
  - PPP
  - Bridge
  - Config
  - SNMP
  - INET
  - Token
  - Ether
  - Radius
  - RIP
  - Frame Relay
  - Modem
  - TAPI

System Messages

ISDN

(**biboAdmSyslogSubject** = **`isdn`**)

biboAdmSyslogMessage	*~Level*
slot <*slot*>, unit <*unit*>, chan <*chan*>: modem connect ... A modem connect occurred for the ISDN device with the given slot, unit and b-channel. The connect baud rate will also be given.	debug
stack <*stkno*>: physical disconnect A physical disconnect occurred for the given ISDN stack. This event occurs due to Power Source 1 (PS1) being lost, for example when the ISDN plug is removed from the ISDN socket.	debug
stack <*stkno*>: activate The layer 1 of the given ISDN stack has been activated, reached the state F7 and is now ready for communication.	debug
stack <*stkno*>: deactivate The layer 1 of the given ISDN stack has been deactivated.	debug
stack <*stkno*>: TEI assign <stkno> The given automatic TEI has been assigned to the given ISDN stack by the network.	debug
stack <*stkno*>: TEI remove The automatic TEI has been removed from the given ISDN stack, because a TEI-REMOVE message has been received from the network	debug
stack <*stkno*>: MDL_ERROR ... The given MDL ERROR (management data link error) has been occurred on the given ISDN stack.	debug

biboAdmSyslogMessage	*~Level*
stack <*stkno*>: disconnect cause: (0x...) A call disconnect occurred on the given ISDN stack with the given cause (See Appendix C: ISDN Error Codes).	debug
stack <*stkno*>: AT&T 5ESS SPID '...' negotiation succeeded	info
stack <*stkno*>: NI-1 SPID '...' negotiation succeeded	info
stack <*stkno*>: DMS-100 SPID '...' negotiation succeeded SPID negotiation succeeded on the given ISDN switch with the given SPID.	info
isdnlogind: receive call from <*number*> si: <*ind*> ai: <*info*> chi 0x... An incoming call has been received by the ISDNLOGIN service from the given ISDN number. The service indicator (<*ind*>) and additional information (<*info*>) values (1TR6) are given as well as the selected ISDN channel.	info
isdnlogind: ignoring call from <*number*> - no matching isdnloginAllowTable entry An incoming ISDN call dispatched for the ISDNLOGIN service was ignored, because the ***isdnLoginAllowTable*** is not empty and does not contain an entry for the calling ISDN number.	info
isdnlogind: accept call from <number> An ISDN call from the given ISDN address was accepted by the ISDNLOGIN service.	info
stack <*stkno*>: q931: mandatory information element missing A call control message has been received with a missing mandatory information element.	err
stack <*stkno*>: AT&T 5ESS SPID '<*spid*>' wrong -> restricted service The given SPID configured in the ***isdnStackTable*** is not valid for a AT&T 5ESS ISDN switch. The ISDN service will be restricted	err

biboAdmSyslogMessage	*~Level*
stack <*stkno*>: NI-1 SPID '<*spid*>' negotiation failed	err
stack <*stkno*>: DMS-100 SPID '<*spid*>' negotiation failed SPID negotiation failed on the given ISDN switch with the given SPID.	err

IPX

(**biboAdmSyslogSubject** = **`ipx`**)

biboAdmSyslogMessage	*~Level*
RIP/SAP: low memory	crit
RIP/SAP: no ipxAdminTable	crit
RIP/SAP: no ipxAdvSysTable	crit
RIP/SAP: no IPX license	info
config error: equal internal netnums	err
no common routing protocol	err
exchange failed: too many retries	err
link down during parameter exchange	err
parameter exchange timed out	err
NAK received, check ipxBasicSysTable	err
unknown packet type <*typeno*> received	err
circuit <*cirxinx*>'s net number not set	err
open link for pkt. type <*typeno*>, dest. socket <*socketno*>	debug
Internal Netnumber not set	err

biboAdmSyslogMessage	*~Level*
Internal Netnumber <*netno*> already in use	err
circuit <*circinx*>'s netnumber <*netno*> already in use	err
ipxDestTable has changed	debug
ipxDestServTable has changed	debug
invalid CircIndex <*circinx*> for static route	err
invalid CircIndex <*circinx*> for static service	err
remote router to net <*netno*> does not support SPX spoofing	info
re-broadcasting NetBIOS packet	debug

CAPI

(**biboAdmSyslogSubject** = **`capi`**)

biboAdmSyslogMessage	*~Level*
got too long TCP message	err
got too long CAPI message	err
got unknown CAPI primitive <*primno*>	err
CAPI message <*primno*> too short, len <*cnt*> should be <*cnt*>	err
CAPI message <*primno*> STRUCT too short, len <*cnt*> should be <*cnt*>	err
APPL <*applno*> PLCI <*plcino*> NCCI <*nccino*> CMD <*cmdno*> CAPIINFO <*infono*>	debug

PPP

(**biboAdmSyslogSubject** = **`ppp`**)

biboAdmSyslogMessage	*~Level*
Channel Delivery Messages:	
request to drop one channel, stack: <*stkno*>	debug
drop one channel, stack: <*stkno*>	debug
Login Procedure Messages:	
login, send: <*string*>	debug
login, expected sequence (<*string*>) received	debug
login, rcvd: <*string*>	debug
PPP/Multi-link PPP Messages:	
can't join non-MP link in non-MP mode can't join MP link in non-MP mode	debug
packet overflow ! to many not currently reassembled MLP fragments received	err
PPP keep alive failed	err
Establishing/Closing Connection Messages:	
no outgoing dial entry	debug
specified isdn hardware not available The specified ISDN hardware was not available.	err
no matching dispatch table entry A matching ***isdnDispatchTable*** entry was not found.	debug
no matching screening indicator	err
dial number <*called number*>	debug
dialin from <*calling number*> to local number <*local number*>	debug

biboAdmSyslogMessage	***~Level***
callback or call collision detected	debug
<*ifc*> is blocked, link establishment failure Interface <ifc> is blocked due to link establishment failure	debug
interface <*ifc*> is administratively down Incoming call, cannot be accepted because interface <*ifc*> is currently down.	debug
call accepted, interface <*ifc*>	debug
call accepted, call not identified by number Inband Authentication follows	debug
<*incoming/outgoing*> connection established	debug
call cleared, specified incoming number doesn't match Inband Authentication case, call accepted and WAN partner identified, incoming number differs from correlating dialtable entry.	err
leased line connection closed, duration <*duration*> seconds, <*bytes*> bytes received, <*bytes*> bytes sent Leased line connection was disconnected. Time and transmit/receive statistics are shown.	info
<*incoming/outgoing*> link closed, duration <*duration*> seconds, <*bytes*> bytes received, <*bytes*> bytes sent, <*charging units*> charging units (no AOCE) An incoming (or outgoing) link was closed. Time and transmit/receive statistics are shown.	info
<*maxconn*> connections exceeded	debug
LCP (Link Control Protocol) Messages:	
loopback detected	err
IPCP (Internet Protocol Control Protocol) Messages:	
local IP address is <*IP address*>, remote is <*IP address*>	info

biboAdmSyslogMessage	*~Level*
remote IP address assigned to <*IP address*>	info
VJHC negotiated, maxslotid is <*negotiated max slot ID*> Van Jacobson TCP/IP header compression negotiated successfully.	info
remote rejected mandatory IPCP option <*option*>	debug
LZS Stac Compression Messages:	
no valid license for Stac LZS	debug
no Stac LZS negotiated, maximum bandwidth exceeded	debug
no function module BIANCA/STAC found	err
unsupported boardtype for Stac LZS compression	debug
Cisco compatible Stac LZS packet format	debug
CCP Stac LZS negotiation successful	debug
remote rejected mandatory CCP option <option>	debug
reset decompression history (<*history number*>)	debug
PAP (Password Authentication Protocol) Messages:	
PAP auth failed for <*id*>	err
PAP auth failed: remote rejected ident/secret	err
PAP auth failed: too many retries	err
CHAP (Challenge Handshake Authentication Protocol) Messages:	
CHAP auth failed for <*id*>	err
CHAP auth rcvd failure: <*failure*>	err
CHAP auth failed: too many retries	err
authentication failed completely	notice

biboAdmSyslogMessage	*~Level*
PAP and CHAP Messages:	
call identified for host <*id*>	debug
no matching PPP entry found for host <*host*>	warning
PAP/CHAP authentication failure	err
Callback Messages:	
incoming call cleared, PPP Callback not enabled Incoming authentication, callback requested by partner via LCP, but not configured.	err
use configured dial number for callback	debug
use negotiated dial number for callback	debug
incoming call cleared	debug
callback follows in <callback delay> seconds Delayed callback.	debug
callback follows at once	debug
incoming call cleared, callback initiated CLID, callback initiated.	debug
Shorthold Messages:	
shorthold timeout reached Static shorthold timeout reached.	debug
dynamic shorthold, <*sec*> seconds after last advice of charge Dynamic shorthold timeout reached.	debug

Bridge

(**biboAdmSyslogSubject** = **`bridge`**)

biboAdmSyslogMessage	*~Level*
no license	info
no mem	err
dialup <*ifc*>	debug
sent TCN thru port <*ifc*> in state <*stateno*>	debug
received CFG thru port <*ifc*> in state <*stateno*>	debug
CFG supersedes port <*ifc*>	debug
received TCN in state <*stateno*>	debug
configuration timed out on ifindex <*ifc*>	debug
TCN timer expired	debug

Config

(**biboAdmSyslogSubject** = **`config`**)

biboAdmSyslogMessage	*~Level*
flash error	crit
tftp: <*error message*>	err
tftp: wrong line <*lineno*> in file <*filename*>	err
unknown object	err
unknown table	err

SNMP

(**biboAdmSyslogSubject** = **`snmp`**)

biboAdmSyslogMessage	*~Level*
no mem available	err

biboAdmSyslogMessage	*~Level*
sent TRAP(<*type*>,<*no*>) <*cnt*> bytes to <*ipno*> Port <*portno*>	debug
sent TRAP(<*type*>,<no>) <*cnt*> bytes to circindex <*no*> Port <*portno*>	debug
snmpInASNParseErr from <*ipno*> Port <*portno*>	debug
received error: <*errmsg*> from <*ipno*> Port <*portno*>	debug
send error: <*errmsg*> to <*ipno*> Port <*portno*>	debug
snmpInBadVersion from <*ipno*>Port <*portno*>	debug

INET

(**biboAdmSyslogSubject** = **`inet`**)

biboAdmSyslogMessage	*~Level*
dialup ifc <*ifc*> prot <*proto*> <*SrcIP*>:<*Port#*> -> <*DestIP*>:<*Port#*> ... The given interface is dynamically dialed up, because a packet has to be routed to it. The protocol of the packet (1=ICMP, 6=TCP, 17=UDP, ...) The source IP address : port number. The destination IP address : port number. If enabled in ***ipExtIfAccessReport***, a dump of the packet also follows.	debug

biboAdmSyslogMessage	*~Level*
NAT: delete session on ifc <*ifc*> prot <*proto*> <*Int*>:<*Port#*>/<*Ext*>:<*Port#*> <-> <*Rem IP Addr*>:<*Port#*> A NAT session is deleted on the given interface. The additional info is: Protocol (1=ICMP, 6=TCP, 17=UDP, ...) . Internal local IP address : portnumber . External local IP address : portnumber . Remote IP address : portnumber.	debug
NAT: new outgoing session on ifc <*ifc*> prot <*proto*> <*Int*>:<*Port#*>/<*Ext*>:<*Port#*> -> <*Rem IP Addr*>:<*Port#*> A new outgoing NAT session is created on the given interface. The additional info is: Protocol (1=ICMP, 6=TCP, 17=UDP, ...). System Interface. Internal local IP address : portnumber . External local IP address : portnumber . Remote IP address : portnumber.	debug
NAT: new session on ifc <*ifc*> prot <*proto*> <*Int*>:<*Port#*>/<*Ext*>:<*Port#*> -> <*Rem IP Addr*>:<*Port#*> A new NAT session is created on the given interface. The additional info is as shown in the previous message.	debug
NAT: new expected session on ifc <*ifc*> prot <*proto*> <*Int*>:<*Port#*>/<*Ext*>:<*Port#*> -> <*Rem IP Addr*>:<*Port#*> A new incoming NAT session is created on the given interface. This session was expected due to an existing IP session. For example, an FTP data session will be expected, when the corresponding portnumbers have been exchanged on a FTP control session The additional info is: Protocol (1=ICMP, 6=TCP, 17=UDP, ...) Internal local IP address : portnumber External local IP address : portnumber Remote IP address : portnumber	debug

biboAdmSyslogMessage	*~Level*
NAT: new incoming session on ifc <*ifc*> prot <*proto*> <*Int*>:<*Port#*>/<*Ext*>:<*Port#*> -> <*Rem IP Addr*>:<*Port#*> A new incoming NAT session is created on the given interface. The session is preconfigured in the ***ipNatPresetTable***. The additional info is: Protocol (1=ICMP, 6=TCP, 17=UDP, ...). Internal local IP address : portnumber. External local IP address : portnumber. Remote IP address : portnumber.	debug
refuse from ifc <*ifc*> prot <*ptoto*> <*Src IP*>:<*Port#*> -> <*Dest IP*>:<*Port#*> An IP packet is being refused due to packet filtering with acces lists. The packet has been received from the given interface. The additinal info is: The protocol of the packet (1=ICMP, 6=TCP, 17=UDP, ...). The source IP address : port number. The destination IP address : port number.	info
NAT: refused incoming session on ifc <*ifc*> prot <*ptoto*> <*Ext IP*>:<*Port#*> <- <*Rem IP*>:<*Port#*> ... An incoming session in the given interface has been refused, because it was neither expected, nor preconfigured in the ***ipNatPresetTable***. The additional info is: Protocol (1=ICMP, 6=TCP, 17=UDP, ...) External local IP address : portnumber Remote IP address : portnumber A dump of the message may follow, if enabled in ***ipExtIfAccessReport*** for the interface .	info
cannot use undefined ifc <*ifc*> for routing The given target interface is configured in the ipRouteTable or the ipExtRouteTable. This interface does not exist and can thus not be used for routing.	err

biboAdmSyslogMessage	*~Level*
cannot use ifc <*ifc*> for routing (ifc does not support IP) The given target interface is configured in the ***ipRouteTable*** or the ***ipExtRouteTable***. This interface does not support IP and can thus not be used for routing	err
NAT: no ipaddress defined on ifc <*ifc*> There is no IP Address defined for the given interface, and NAT has been enabled. An interface's IP address must be defined for NAT to work.	err

Token

(**biboAdmSyslogSubject** = **`token`**)

biboAdmSyslogMessage	*~Level*
slot <*slot#*>: BUD Error SIFCMD=<*command*>	err
slot <*slot#*>: Hardware Error	err
slot <*slot#*>: DMA test failed	err
slot <*slot#*>: Initialization Error Code=...	err
slot <*slot#*>: Open failed (... in ...)	err
slot <*slot#*>: Ring status = <*state*>	err
TMS380SRA diagnostics failed	err
TMS380SRA does not exist	err
SRA bad options	err
SRA error parm0=0x...	err
slot <*slot#*>: Adapter check sts=0x... parm0=0x... parm1=0x... parm2=0x...	err
slot <*slot#*>: cmd reject rej_sts=0x... rej_cmd=0x...	err

biboAdmSyslogMessage	*~Level*
slot <*slot#*>: Open Error code 0x...	err
slot <*slot#*>: read errlog failed	err
slot <*slot#*>: read adapter failed	err
slot <*slot#*>: modify open parms failed	err
slot <*slot#*>: unknown cmd (cmd=0x... parm0=0x... parm1=0x...)	err
slot <*slot#*>: receive suspended	err
slot <*slot#*>: transmit list error 0x...	err
slot <*slot#*>: unexpected interrupt <*int*>	err
slot <*slot#*>: Ring insertion succeeded	debug
slot <*slot#*>: Adapter closed	debug
slot <*slot#*>: Ring status = <*state*>	debug

Ether

(**biboAdmSyslogSubject** = **`Ether`**)

biboAdmSyslogMessage	*~Level*
slot <*slot#*>: out of sync ... ... sts=.../... size=.../...\n Internal synchronization problem on the ethernet controller in the specified slot. Automatic recovery occurs. The additional information is of no use for the user.	warning
slot <*slot#*>: Excessive Deferral (Transmission aborted) - Cable Problem? The given ethernet controller reports "excessive deferal" on the network. This may be due to a cable problem.	warning

biboAdmSyslogMessage	*~Level*
slot <*slot#*>: No Carrier Sense - Cable Problem ? The given ethernet controller reports "no carrier sense" on the network. This may be due to a cable problem.	warning
slot <*slot#*>: CD Heartbeat lost The heartbeat signal between the transceiver and the Ethernet controller did not occur for the given slot. This is usually the case, when an external transceiver is connected to the AUI-interface and not support the heartbeat signal, or the heartbeat signal is switched off on the external transceiver.	warning
slot <*slot#*>: sonic smuttier hung, resetting The transmitter of the ethernet controller stopped working and is reset automatically.	warning
slot <*slot#*>: Excessive Collisions (Transmission aborted) The given ethernet controller reports "excessive collision". The transmission of the current packet is aborted and continued with the next packet. This warning may occur due to very high network load, to problems with the network itself or cabling problems.	debug

Radius

(**biboAdmSyslogSubject** = **`radius`**)

biboAdmSyslogMessage	*~Level*
Inband RADIUS Messages:	
cannot accept call for Radius client <id>	debug
call identified for Radius client <id>	debug
Radius PAP auth failed for <id>	notice
Radius CHAP auth failed for <id>	notice

biboAdmSyslogMessage	*~Level*
Outband RADIUS Messages:	
outband identification ok	debug
outband identification failed, try inband	debug
outband identification timed out, try inband	debug

RIP

(**biboAdmSyslogSubject** = **`rip`**)

biboAdmSyslogMessage	*~Level*
ROUTE ADD ifc <*ifc*> Dest <*dest_addr*> Mask <*netmask*> Metric <*metric*> Nexthop <*ip_address*> Age <*age*>	debug
ROUTE DEL ifc <*ifc*> Dest <*dest_addr*> Mask <*netmask*> Metric <*metric*> Nexthop <*ip_address*> Age <*age*>	debug
ROUTE CHANGE ifc <*ifc*> Dest <*dest_addr*> Mask <*mask*> Metric <*metric*> Nexthop <*ip_address*> Age <*age*>	debug

Frame Relay

(**biboAdmSyslogSubject** = **`fr`**)

biboAdmSyslogMessage	*~Level*
Be exceeded - packet discarded unknown ARP protocol <*proto*>	debug
no license	info
no more than 256 interfaces allowed	error
DLCI out of range: <*dlci*>	notice

Modem

(**biboAdmSyslogSubject** = **`modem`**)

biboAdmSyslogMessage	*~Level*
no more modems available <*inuse*>/<*registered*>/ <*maxmodems*>	debug

TAPI

(**biboAdmSyslogSubject** = **`tapi`**)

biboAdmSyslogMessage	*~Level*
no license	info

GLOSSARY OF NETWORKING TERMS

The networking field is pockmarked with acronyms that are often used inconsistently throughout the trade. Following is a brief glossary of some of the terms used within

10BaseT – An IEEE standard (802.3) for operating 19 Mbps Ethernet networks with twisted pair cabling and a wiring hub. See also UTP.

1TR6 – An ISDN D-channel protocol that was used in Germany prior to the widespread implementation of the DSS1. Currently 1TR6 is being replaced by the DSS1 protocol.

ARP (Address Resolution Protocol) – The protocol in the TCP/IP suite that is used to obtain the network point of attachment address (usually the MAC or ethernet address) of a host corresponding to it's internet address.

AUI (Autonomous Unit Interface) – Also called an Attachment Unit Interface. This refers to the 15 pin D connector and cables that connect single and multiple channel equipment in an Ethernet transceiver.

Address Resolution Protocol – See ARP.

Agent – The client-server model, the part of the system that performs information preparation and exchange on behalf of a client or server application.

Autonomous Unit Interface – See AUI.

B-Channel – ISDN bearer service channel operating at 64 kbps, carrying user voice or data; circuit-, packet-, or frame-mode services may be obtained on this channel.

BRI (Basic Rate Interface) – One of the access methods to an ISDN, comprising two B-channels and one D-channel (often referred to as 2B+D).

Bandwidth – The width of a channel's passband (e.g., the bandwidth of a channel with a 300- to 3400-Hz passband is 3100 Hz, or 3.1 kHz).

Basic Rate Interface – See BRI.

Bearer service – The basic set of services offered over the B-channel that provides the capability to exchange signals between two user-network interfaces.

BootP – The Bootstrap Protocol is a UDP/IP-based protocol which allows a booting host to configure itself dynamically and without user supervision.

Bridge – Bridges can usually be made to filter packets, that is, to forward only certain traffic. Related devices are: repeaters which simply forward electrical signals from one cable to another, and full-fledged routers which make routing decisions based on several criteria.

Broadcast – A means of transmitting a message to all devices connected to a network. Normally, a special address, the broadcast address, is reserved to enable all the devices to determine that the message is a broadcast message.

Bus – A network transmission medium to which all the devices are attached. Each transmission propagates the length of the medium and is therefore received by all other devices connected to the medium.

CAPI (Common ISDN Application Programming Interface) – An application programming interface standard resulting from close cooperation with leading ISDN manufacturers. CAPI defines the entity and the protocol that applications must use when communicating with this entity.

CGI (Common Gateway Interface) – A standard for running external programs from a World-Wide Web (HTTP) server. CGI specifies how to pass arguments to the executing program as part of the HTTP request. Common-

ly, the program will generate an HTML document which will be passed back to the browser but it can also request redirection to a different document.

CCIT (Intergraph and Telephone Consultative Committee) – A committee of the ITU, creating recommendations regarding public telegraph, telephone, and data networks. Renamed ITSS in March 1993.

CHAP (Challenge Handshake Authentication Protocol) – Under PPP, each system may require it's peer to authenticate itself using the CHAP protocol or the PAP protocol.

CLID (Calling Line ID) – A telephone company service that delivers the calling party's telephone number to the called party during the ring cycle; also called "automatic number identification".

CRC (Cyclic Redundancy Check) – A method used for the detection of errors when data is being transmitted. A CRC is a numeric value computed from the bits in the message to be transmitted. It is appended to the tail of the message prior to transmission and the receiver then detects the presence of errors in the received message by computing a new CRC.

CSMA/CD – An abbreviation for carrier sense, multiple access with collision detection. It is a method used to control access to a shared transmission medium such as coaxial cable bus to which a number of stations are connected. A station wishing to transmit a message first senses (listens) the medium and transmits a message only if the medium is quiet—no carrier present. Then, as the message is being transmitted the station monitors the actual signal on the transmission medium. If this is different from the signal being transmitted, a collision is said to have occurred and been detected. The station ceases transmission and retries again later.

Calling Line ID – See CLID.

Challenge Handshake Authentication Protocol – See CHAP.

Coaxial Cable – A type of transmission medium consisting of a center conductor and a cocentric outer conductor. It is used when higher data transfer rates (greater than 1 Mbps) are required.

Common Gateway Interface – See CGI.

Common ISDN Application Programming Interface – See CAPI.

Cyclic Redundancy Check – See CRC.

DCE (Data Circuit-terminating Equipment) – The name given to the equipment provided by the network authority (provider) for the attachment of user devices to the network. It takes on different forms for different types of networks.

DHCP (Dynamic Host Configuration Protocol) – A protocol introduced by Microsoft. The protocol provides a means to dynamically allocate IP addresses (and other network information) to PCs running on a Microsoft Windows local area network. The system administrator assigns a range of addresses to a DHCP server and each PC is configured to request its IP address from the server. The request and grant process uses a lease concept with an adjustable time period.

DLCI (Data Link Connnection Identifier) – In a Frame Relay network, a DLCI uniquely identifies a single virtual circuit. It is important to note that a DLCI is only significant to the local side of a point-to-point link.

DSS1 (Digital Subscriber Signalling System) – The ISDN user-network interface, comprising a data link layer and network layer; described in CCITT (now ITU) Recommendations Q.920-series (LAPD/LAPF) and Q.930-series recommendations, respectively.

DTE (Data terminal equipment) – A generic name for any user device connected to a data network. It thus includes such devices as visual displays, computers, and office workstations.

D-Channel – The ISDN out-of-band signalling channel, carrying ISDN user-network messages; it can also be used to carry packet- or frame-mode user data. The D-channel operates at 16 kbps in the BRI and 64 kbps in the PRI.

Data Circuit-terminating Equipment – See DCE.

Data Link Connnection Identifier – See DLCI.

Data Link Layer – It is concerned with the reliable transfer of data (no residual transmission errors) across a data link being used.

Data Link Connection Identifier – See DLCI.

Data terminal equipment – See DTE.

Datagram – A self-contained packet of information that is sent through the network with minimum protocol overheads.

Digital Subscriber Signalling System – See DSS1.

Domain – In the Internet, a part of a naming hierarchy. Syntactically, an Internet domain name consists of a sequence of names (labels) separated by periods (dots), e.g., "tundra.mpk.ca.us." In OSI, "domain" is generally used as an administrative partition of a complex distributed system, as in MHS Private Management Domain (PRMD), and Directory Management Domain (DMD).

Dotted decimal notation – The syntactic representation for a 32-bit integer that consists of four 8-bit numbers written in base 10 with periods (dots) separating them. Used to represent IP addresses in the Internet as in: 192.67.67.20

Dynamic host configuration protocol – See DHCP.

EAZ (Endgeräteauswahlziffer) – In the 1TR6 protocol, the last digit of the ISDN number, which combined with the service indicator allows a specific end station to be identified.

ET (Exchange Termination) – That portion of the local exchange that assumes the responsibility for LE's communication with the other network components of the ISDN.

ETSI (European Telecommunications Standards Institute) – An organization, headquartered in France, responsible for creating common telecommunications standards for the European market.

Ethernet – A local area network that connects devices (computers, printers, etc.) via twisted pair or coaxial cabling.

Encapsulation – A technique used by layered protocols in which a layer adds header information to the protocol data unit (PDU) from the layer above. As an example, in Internet terminology, a packet would contain a header from the physical layer, followed by a header from the network layer (IP), followed by a header from the transport layer (TCP), followed by the application protocol data.

Endgeräteauswahlziffer – See EAZ.

European Telecommunications Standards Institute – See ETSI.

Exchange Termination – See ET.

FCS (Frame check sequence) – A general term given to the bits appended to a transmitted frame or message by the source to enable the receiver to detect possible transmission errors.

FTP (File Transfer Protocol) – The TCP/IP protocol (and program) used to transfer files between hosts.

File Transfer Protocol – See FTP.

Filter – A rule that defines a set of packets. Filters can be used to specify a set of packets that may or may-not-be routed.

Firewall – A mechanism consisting of hardware and/or software that let's an administrator control the types of packets may access the network (pass through a router).

Fragmentation – The process in which an IP datagram is broken into smaller pieces to fit the requirements of a given physical network. The reverse process is termed reassembly. Also see MTU.

Frame check sequence – See FCS.

Frame – The unit of information transferred across a data link.

Frame Relay – A form of packet switching that uses smaller packets and less error checking than traditional packet switching such as X.25. Due to these characteristics Frame Relay is effective for handling high-speed, bursty traffic over Wide Area Networks.

Full Duplex – Bidirectional communications facility where transmissions may travel in both directions simultaneously. Also called duplex.

Gateway – The original Internet term for what is now called router or more precisely, IP router. In modern usage, the terms "gateway" and "application gateway" refer to systems which do translation from some native format to another. Examples include X.400 to/from RFC 822 electronic mail gateways. See router.

HTTP (HyperText Transfer Protocol) – The TCP/IP protocol used on the World-Wide Web for the exchange of HTML documents between client and server systems. It conventionally uses TCP port 80.

HDLC (High level data link control) – An internationally agreed standard protocol defined to control the exchange of data across either a PPP data link or a multidrop data link.

Half Duplex – Bidirectional communications facility where transmissions may travel in either one direction or the other at any given time. Sometimes referred to as simplex, outside on North America.

High level data link control – See HDLC.

Host – This is normally a computer that contains (hosts) the communication hardware necessary to connect the computer belonging to a data communication network.

Hypertext Transfer Protocol – See HTTP.

ICMP (Internet Control Message Protocol) – The protocol used to handle errors and control messages at the IP layer. ICMP is actually part of the IP protocol.

IGP (Interior Gateway Protocol) –

IP (Internet Protocol) – The network layer protocol for the Internet protocol suite.

IP datagram – The fundamental unit of information passed across the Internet. Contains source and destination addresses along with data and a number of fields which define such things as the length of the datagram, the

header checksum, and flags to say whether the datagram can be (or has been) fragmented.

IPX (Internetwork Packet exchange) – A network layer protocol initially developed at XEROX Corporation and made popular by Novell, Inc. It is the basic protocol in Novell NetWare's file server operating system and allows Novell clients and servers to communication over LAN/WAN links.

ISDN (Integrated Services Digital Network) – A technology which combines, or "integrates", various services including telephony, telex, data transfer, fax, teletex, and videotex in a single digital medium. ISDN makes it possible for customers to access all of these digital data services through a single "wire." The standards that define ISDN are specified by ITU.

ISO (International Organization for Standardization) – An international standards organization that comprises national standards bodies; ANSI, for example, is the U.S. representative to ISO.

ISP (Internet Service Provider) – A company which provides other companies or individuals with access to, or presence on, the Internet.

ISDN address – An address of a specific ISDN device; comprises an ISDN number plus additional digits that identify a specific terminal at a user's interface (e.g. 47117).

ISDN number – The network address associated with a user's ISDN interface (e.g. 4711).

ITU (International Telecommunication Union) – An agency of the United Nations, the parent organization of the CCITT (now called the ITSS).

Integrated Services Digital Network – See ISDN.

Interior Gateway Protocol (IGP) – See IGP.

International Organization for Standardization – See ISO.

International Telecommunication Union – See ITU.

Intergraph and Telephone Consultative Committee – See CCIT.

Internet Control Message Protocol – See ICMP.

Internet (with a capital "I") **–** The largest internet consisting of large national backbone nets (such as MILNET, NSFNET, and CREN) and a myriad of regional and local campus networks worldwide. The Internet uses the Internet protocol suite. To be on the Internet you must have IP connectivity, i.e., be able to Telnet to--or ping--other systems. Networks with only e-mail connectivity are not actually classified as being on the Internet.

Internet Protocol – See IP.

Internet Service Provider – See ISP.

ISDN (Integrated Services Digital Network) – A technology which combines, or "integrates", various services including telephony, telex, data transfer, fax, teletex, and videotex in a single digital medium. ISDN makes it possible for customers to access all of these digital data services through a single "wire." The standards that define ISDN are specified by ITU.

LAPB (Link Access Procedure Balanced) – The X.25 data link layer protocol.

LCP (Link Control Protocol) – A protocol used by PPP to automatically agree upon encapsulation format options, handle varying packet size limits, authenticate the identity of its peer on the the link, determine when a link is functioning properly and when it is defunct, detect common misconfiguration errors, and terminate the link. See RFC 1570.

LE (Local Exchange) – An ISDN central office.

LLC (Link Layer Control) – The upper portion of the data link layer, as defined in IEEE 802.2. The LLC sublayer presents a uniform interface to the user of the data link service, usually the network layer. Beneath the LLC sublayer is the Media Access Control (MAC) sublayer.

LT (Local Termination) – That portion of the local exchange responsible for functions related to the termination of the local loop.

Link Access Procedure Balanced – See LAPB.

Link Access Procedure on the D-channel – The ISDN data link layer protocol specified for the D-channel.

Link Control Protocol – See LCP.

Link Layer Control – See LLC.

Local Exchange – See LE.

Local Termination – See LT.

MIB (Management Information Base) – A collection of objects that can be accessed via a network management protocol. See SMI.

MTU (Maximum Transmission Unit) – The largest possible unit of data that can be sent on a given physical medium. Example:
The MTU of Ethernet is 1500 bytes. See fragmentation.

MAC (Medium access control) – Many local area networks utilize a single transmission medium – a bus, or ring for example. to which all the connected devices are attached. A procedure must be followed for each device to ensure that transmissions occur in an orderly manner. In general, this is known as the medium access control procedure. Two examples are CSMA/CD and token ring.

MSN (Multiple Subscriber Number) – In Q.931 compatible D-channel protocols, multiple telephone numbers can be used to establish a connection with a single endpoint. Using these MSNs and an appropriate service indicator a specific piece of terminal equipment or a service provided by that equipment can be identified.

Management Information Base – See MIB.

Maximum Transmission Unit – See MTU.

Medium access control – See MAC.

Modem – (Modulator/demodulator) An electronic device (DCE) typically used for converting serial data between computing equipment (DTE) and and an analog transmission channel such as a phone line.

Multi-homed host – A computer in an IP network that is connected to more than one interface can have more than one IP address (or MAC ad-

dress). Such a host can be called a “multi-homed” host. The interfaces may or may not be attached to the same network.

Multicast – A special form of broadcast where copies of the packet are delivered to only a subset of all possible destinations. See broadcast.

Multiple Subscriber Number – See MSN.

NMS (Network Management Station) – The system responsible for managing a (portion of a) network. The NMS talks to network management agents, which reside in the managed nodes, via a network management protocol. See also: Agent, SNMP.

NAT (Network Address Translation) – (Sometimes called Virtual LAN) A software mechanism (provided by an IP router) that allows one to extend the Internet address already in use. IP addresses used on a LAN are "translated" to differed address when packets traverse the translating device.

NSAP (Network Service Access Point) – NSAP is an alternative addressing scheme used in a few X.25 data networks. The format of an NSAP address is defined in the X.213 recommendation and includes both OSI-conformant and non OSI-conformant versions.

NT1 (Network Termination Type1) – The ISDN device responsible for the termination of the ISDN transmission facility at the customer premises.

NT2 (Network Termination Type2) – An ISDN device responsible for on-premises communication distribution, such as a PBX, LAN, or host computer.

NetBEUI – NetBIOS Extended User Interface. The network transport protocol used by all of Microsoft’s network systems and IBM’s LAN Server based systems. NetBEUI is often confused with NetBIOS. NetBIOS is the applications programming interface and NetBEUI is the transport protocol.

NetBIOS – (Note: BIOS from Basic Input Output System) An applications programming interface (API) which activates network operations on a PC running under Microsoft’s DOS. It is a set of commands that the application program issues in order totransmit and receive data to another host on the

network. The commands are interpreted by a network control program or network operating system.

Network Address Translation – See NAT.

Network Management Station – See NMS.

Network Termination Type1 – See NT1.

Network Termination Type2 – See NT2.

OSPF (Open Shortest Path First) – One of the Internet standard Interior Gateway Protocols (IGP) defined in RFC 1247. OSPF is a link state routing protocol, as opposed to a distance vector routing protocol (used by RIP, the most common IGP).

Octet – Eight data bits.

PABX (Private Automatic Branch eXchange) – An automatic PBX.

PAP (Protocol Authentication Protocol) – Under PPP, each system require it's peeMr to authenticate itself using either PAP or CHAP.

PBX (Private Branch exchange) – A customer site telephone switch. Common usage of this term today implies that a PBX is an automatic switch, although a PBX could be under the control on an operator (or attendant).

PDF (Portable Document Format) – The native file format for Adobe Systems' Acrobat. PDF is the file format for representing documents in a manner independent of the original application software, hardware, and operating system used to create those documents.

PH (Packet Handler) – A packet switch (or X.25 DCE equivalent device) in an ISDN.

POTS (Plain Old Telephone Service) – The plain old telephone service is a reference to the traditional analog telephone system.

PPP (Point-to-Point Protocol) – The successor to SLIP, PPP provides router-to-router and host-to-network connections over both synchronous and asynchronous circuits.

PRI (Primary Rate Interface) – (a.k.a T1 PRI Line in USA) An ISDN PRI interface consists of a D-channel for signalling and 23 (USA) or 30 (Europe) B-channels for user data. The B-channels may be switched or combined depending on services from the local provider.

PSN (Packet Switched Network) – A data communications network using packet switching technology; commonly supports the X.25 interface.

PSTN (Public Switched Telephone Network) – The public switched telephone network is just another term for the analog telephone system.

Packet Handler – See PH.

Packet Switched Network – See PSN.

Packet switching – A switching procedure whereby two parties have a logical connection across a network, but no dedicated facilities, and where units of transmission are variable in length but have a maximum size. This is a store-and-forward technique where nodes in the network may store a packet for some time before forwarding it to the next node in line.

Ping (Packet INternet Groper) – A program used to test reachability of destinations by sending them an ICMP echo request and waiting for a reply. The term is used as a verb**:** "Ping host X to see if it is up!"

Point-to-Point Protocol – See PPP.

Point-to-multipoint ISDN Configuration – A physical connection in which a single network termination supports multiple terminal equipment devices; only supported by the BRI.

Point-to-point ISDN Configuration – A physical connection in which a single network termination supports one terminal equipment device; supported by the BRI or PRI.

Port – The abstraction used by Internet transport protocols to distinguish among multiple simultaneous connections to a single destination host. See selector.

Primary Rate Interface – See PRI.

Private Automatic Branch exchange – See PABX.

Private Branch exchange – See PBX.

Protocol Authentication Protocol – See PAP.

Protocol – A formal description of messages to be exchanged and rules to be followed for two or more systems to exchange information.

Public Switched Telephone Network – See PSTN.

Q.930 – A CCITT recommendation describing the general aspects of the D-channel level 3 protocol; also called recommendation I.450; the Q.930 series recommendations form the DSS1 network layer.

RADIUS (Remote Dial In User Service) – A Client-Server based security system often used by Internet ServiceProviders (ISPs). RADIUS defines a mechanism by which dial-in users can be granted (or denied) access to network services using a centrally managed server that exchanges authentication information (usually UDP/IP) about the user with a RADIUS client.

RARP (Reverse Address Resolution Protocol) – For hosts that can't store their IP address locally (diskless workstations) RARP is often used. When such a workstation comes into service it asks for its IP address by broadcasting a RARP-request that contains its own hardware address. A RARP server usually responds by replying with the IP/MAC address pair of the workstation. (Also see ARP).

Remote CAPI – Remote CAPI is a client-server system that allows CAPI applications running on any PC (where Remote CAPI is is installed) to utilize the ISDN interfaces of a BRICK. The remote CAPI client (Windows dll) forwards all CAPI messages to the BRICK via a TCP data stream. The Remote CAPI server (capid process on the BRICK) forwards all CAPI messages to connected clients via a TCP stream.

Remote TAPI – Remote TAPI is a client-server system that allows TAPI applications running on a PC to access the telephony functionality of a BRICK. The remote TAPI client (Windows dll) forwards all TAPI messages to the BRICK via a TCP data stream. The Remote TAPI server (the tapid on the

BRICK) forwards all TAPI messages to connected clients via a TCP stream. See also TAPI.

Repeater – A device which propagates electrical signals from one cable to another without making routing decisions or providing packet filtering. In OSI terminology, a repeater is a Physical Layer intermediate system. See bridge and router.

Reverse Address Resolution Protocol – See RARP.

RFC (Request For Comments) – The document series, begun in 1969, which describes the Internet suite of protocols and related experiments. Not all (in fact very few) RFCs describe Internet standards, but all Internet standards are written up as RFCs.

RIP (Routing Information Protocol) – An Interior Gateway Protocol (IGP) supplied with Berkeley UNIX. RIP is distance vector algorithm, as opposed to link state, routing protocol. RIP is defined in STD 34, RFC 1058 and updated by RFC 1388.

Request For Comments – See RFC.

Router – A system responsible for making decisions about which of several paths network (or Internet) traffic will follow. To do this it uses a routing protocol to gain information about the network, and algorithms to choose the best route based on several criteria known as “routing metrics.” In OSI terminology, a router is a Network Layer intermediate system. In TCP terminology, a router is often referred to as a gateway. See gateway, bridge and repeater

Routing Information Protocol – See RIP.

SAP (Service Advertising Protocol) – A Novell NetWare protocol that permits file, print, and gateway servers to advertise their services and addresses to other servers and cients.

SAPI (Service Application Identifier) – A subfield in the LAPD address field which carries the type of level 3 service being obtained.

SNMP (Simple Network Management Protocol) – An application protocol in a TCP/IP suite used to send and retrieve management related information across a TCP/IP network. The network management protocol of choice for TCP/IP-based internets.

STAC – An enhanced compression algorithm as defined in RFCs 1974 (*PPP Stac LZS Compression*). The Stacker LZS algorithm was originally developed by Hi/fn, Inc.

SMI (Structure of Management Information) – The rules used to define the objects that can be accessed via a network management protocol. See also MIB.

SPI (Service Provider interface) – In TAPI

SPID (Service Profile Identifier) – SPIDs are used in National ISDN 1 (USA) to identify an ISDN B-channel. Though normally based on your telephone number the format (prefix and suffix digits) and number (one SPID per B-channel, or one for both) of SPIDs depends on your service provider.

SPX (Sequenced Packet Exchange) – SPX is a transport layer protocol used by Novell NetWare systems on top of IPX. See also IPX.

SS7 (Signalling System 7) – The high speed, digital common channel signalling network required for ISDN applications; also provides a myriad of services based on the calling party's ISDN number.

SVC (Switched Virtual Circuit) – A virtual circuit service that is established on demand as needed and relinquished when the data exchange is complete; requires call control procedures for the establishment and termination of the call; SVCs are supported by both X.25 and frame relay.

Service Application Identifier – See SAPI.

Service Indication – Service indication is a part of the ISDN address that describes the type of ISDN service to be used. In DSS1, service indication consists of the BC (bearer capability), HLC (High Layer Compatibility), and LLC (Low Layer Compatibility), elements. In 1TR6, service indication consists of the SI (service indicator), and AI (additional information) elements.

Service Profile Identifier – See SPID.

Signalling System 7 – See SS7.

Spanning Tree Algorithm – An IEEE 802.1 standard (IEEE802.1d-1990) under consideration that provides distributed routing over multiple LANs connected by bridges.

Simple Network Management Protocol – See SNMP.

Structure of Management Information – See SMI.

Subnet – In TCP/IP terminology, a working scheme that divides a single logical network into smaller physical networks to simplify routing.

Subnetwork – A collection of OSI end systems and intermediate systems under the control of a single administrative domain and utilizing a single network access protocol. Examples:
private X.25 networks, collection of bridged LANs.

Switched Virtual Circuit – See SVC.

TA (Terminal Adapter) – A protocol converter used to allow a non-ISDN terminal to access the network using ISDN protocols and procedures.

TAPI (Telephony Applications Programming Interface) – TAPI is a software interface defined by Microsoft and Intel for developing Windows-based telephony applications. TAPI applications can make, accept and monitor calls. The Microsoft Dialer (part of Windows) is an example of a TAPI application. If the Telephony Service Provider (see TSP) supports supplementary services the TAPI application will also be able to redirect, hold, and make conference calls.

TCP (Transmission Control Protocol) – The major transport protocol in the Internet suite of protocols providing reliable, connection- oriented, full-duplex streams. Uses IP for delivery.

TFTP (Trivial File Transfer Protocol) – A simple file transfer protocol often used by diskless workstations to download their boot code. Note: TFTP is implemented on the BRICK and is used to exchange configuration files and upgrade system software.

TEI (Terminal Endpoint Identifier) – A subfield in the LAPD address field that identifies a given TE device on the ISDN interface.

TSP (Telephone Service Provider) – A TSP uses the TSPI (Telephony Service Provider Interface) defined by Microsoft to support TAPI services for a specific piece of hardware. TAPI supports multiple TSPs allowing the end-user to access differnt hardware at the same time.

Telephony Applications Programming Interface – See TAPI.

Telephony Service Provider Interface – See TSP.

Telnet (Telecomunications Network) – The virtual terminal protocol in the Internet suite of protocols. Allows users of one host to log into a remote host and interact as normal terminal users of that host.

Terminal Adapter – See TA.

Terminal Endpoint Identifier – See TEI.

Transceiver/Transmitter-receiver – The physical device that connects a host interface to a local area network, such as Ethernet. Ethernet transceivers contain electronics that apply signals to the cable and sense collisions.

Transmission Control Protocol – See TCP.

Trivial File Transfer Protocol – See TFTP.

Twisted pair – A type of transmission medium consisting of two insulated wires twisted together to improve the immunity to interference from other (stray) electrical signals which might otherwise corrupt the signal being transmitted.

UDP (User Datagram Protocol) – A transport protocol in the Internet suite of protocols. UDP, like TCP, uses IP for delivery; however, unlike TCP, UDP provides for exchange of datagrams without acknowledgements or guaranteed delivery.

UTP (Unshielded Twisted Pair) – A tranmission medium consisting of two insulated wires twisted together to protect it from other electrical signals that might otherwise corrupt the transmitted signal.

UUCP (UNIX to UNIX Copy Program) – A protocol used for communication between consenting UNIX systems.

UNIX to UNIX Copy Program – See UUCP.

User Datagram Protocol – See UDP.

V.110 – A rate adaption scheme to convert asynchronous or synchronous transmission at rates from 50 bps to 19.2 kbps to the B-channel 64kbps rate; limited to only one low-speed device per B-channel; widely used outside of North America; also called recommendation I.465.

V.42 bis – A widely accepted standard that describes a compression procedure used for transmitting data over telephone networks. See also STAC (an alternative compression algorithm).

VC (Virtual Circuit) – In a store-and-forward network, a logical end-to-end connection between two hosts; the VC must be established at service subscription time or on demand by the user, but the network does not dedicate a transmission facility to this connection.

Virtual Circuit – See VC.

X.21 – The X.21 recommendation describes the physical interface between two DTEs in circuit-switched data networks such as Datex-P in Germany.

X.25 – An internationally agreed standard protocol defined for the interface of a data terminal device, such as a computer, to a packet-switched data network.

X.75 – A CCITT recommendation describing layers 1 through 3 of the interface between PSNs, including PSPDNs and ISDNs.