http://www.bintec.de

RELEASE NOTE
BIANCA/BRICK-XM

April 7,1997

New System Software:
Release 4.3 Revision 13

This document describes the new features, enhancements, bug-fixes, and changes to the BIANCA/BRICK-XM System Software since Release 4.2, and is provided as a supplement to your existing documentation.

Upgrading System Software

1. Retrieve the current system software image from BinTec's FTP server at *http://www.bintec.de*.

2. With this image you can upgrade the BIANCA / BRICK-XM with the **update** command from the SNMP shell via a remote host (i.e. using telnet, minipad, or isdnlogin) or by using the **BOOTmonitor** if you are logged in directly on the console.
   Information on using the BOOTmonitor can be found in your *BRICK-XM User's Guide* under *Firmware Upgrades*.

3. Once you've installed Release 4.3 Revision 13, you may retrieve the latest documentation (in Adobe's PDF format) which is also available from BinTec's FTP server noted above.
   Keep this document for future reference.

Note: When upgrading system software, it is also recommended that you use the most current versions of *BRICKware for Windows* and *UNIXTools*. Both can be retrieved from BinTec's FTP server mentioned above.

What's New in Revision 13?

4.3 Revision 13: Released: 07.04.97

Features:

ISDN in Japan

- The BRICK now supports the NTT INS64 ISDN protocol used in Japan.

Bugfixes:

CAPI

- If the size of the incoming CAPI_DATAB3_IND block exceeds the registered b3 datablock length then the blocklength is decreased to the maximum registered length.
- The BRICK now properly transmits a 'user not responding' cause code to the calling CAPI application when an incoming call is ignored by the receiving application.

Ethernet

- When Ethernet-LLC framing was used the BRICK sometimes incorrectly sent an FRMR (FrameReject).

RADIUS

- When used as a RADIUS client the BRICK sometimes rebooted unexpectedly. This has been corrected.

X.25

- Incoming flow control for X.25 interfaces has been corrected in revision 13.

Changes in Previous Revisions

4.3 Revision 12: Released: 25.03.97

Bugfixes:

PPP

- ISDN Callback didn't work properly in revision 11. Revision 12 fixes this and ISDN callback is now reliable.

RADIUS

- When used as a RADIUS client the BRICK sometimes rebooted unexpectedly. Revision 12 corrects this.

X.25

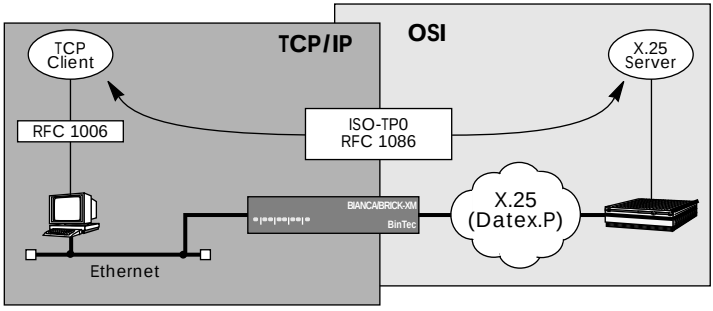
- The special X.25 rewriting rule that allows the X.25 Source Address to be rewritten using the caller's ISDN telno now works reliably. This X.25 enhancement was added in revision 11.
- In previous releases the status of data connections wasn't reliably signalled to higher level X.25 applications such as minipad. This has been corrected in revision 12.

4.3 Revision 11: Released: 11.03.97

Features:

RFC 1086 Support

The BRICK can be effectively used as a TCP-X.25 bridge as described in RFC 1086 *ISO-TP0 bridge between TCP and X.25*. Using this mechanism the BRICK provides an end-to-end ISO-TP0 connection between TCP and X.25 networks as shown below.



Configuration

The requirements for using this mechanism are as follows.

1. The TCP host must support RFC 1006 which defines how to transport TP0 traffic using TCP.
2. The X.25 subsystem must be activated (licensed).
3. X.25 routing is configured so that at a minimum:
   outgoing calls from the special *local* interface are allowed
   all incoming X.25 calls are given to the *local* interface.

These tasks are covered in the *BRICK-XM User's Guide*.

RFC 1086 was supported in previous releases was re-implemented in version 4.3.11. The new implementation provides a performance increase of 100%.

Setup Tool

A new X.25 Monitor has been added to Setup Tool's MONITORING AND DEBUGGING → menu which can be used to monitor current and closed X.25 calls.

IP addresses used for Dynamic IP address assignment can now be configured by defining a range (block) of addresses. See the IP → DYNAMIC IP ADDRESSES → menu.

Remote CAPI for Novell NetWare

The BIANCA / BRICK Companion CD now also contains NLMs which enable you to run CAPI 1.1 and CAPI 2.0 applications on your Novell network utilizing the Recmote CAPI feature of your BRICK-XM.

For further details please refer to README.TXT in the *netware* directory on the CD.

Enhancements:

Configuration File Loading

- In previous releases loading large configuration files (more than 100 partners) from Flash was often slow. In release 4.3.11 the loading procedure was optimized so that large files can be loaded faster.

IPX

- The BRICK now supports IPX connections to remote routers that don't support IPX over WANs according to RFC 1634. Note that the BRICK reverts to standard values for *ipxCircDelay*, *ipxCircNeighRouterName* and *ipxCircType* since negotiating them is not possible with such routers. These routers may need to be configured manually to ensure that only options understood by both are used.

IP

- The BRICKs ping command now offers the new option *-c ‹count›* which lets you specify the number of ping packets to be sent.

RADIUS

- It is now possible to use dynamic IP address assignment in conjunction with RADIUS.

X.25

- A special "#" character has been added to the SrcAddr field of the x25RewriteTable. For incoming calls this character is replaced by the incoming caller's ISDN number. For example, if SrcAddr=88#88, then an incoming call from 777 is rewritten as 8877788.

Bugfixes:

CAPI

- Passive FAX polling using the Layer 3 protocol "T.30 extended" (CAPI 2.0) now works properly.
- A FAX error which occasionally lead to a reboot while receiving a fax has been fixed.
- If an application does not accept a CONNECT_IND from the CAPI, the DISCONNECT_IND message now signals "User not responding" (CAPI 1.1: info=0x34ba; CAPI 2.0: info=0x3492) instead of info=0x3302 (error on setup of d-channel layer3) as before.
- SELECT_B_PROTOCOL_REQ messages weren't reliable. After a FAX (or other transparent) connection and a subsequent SELECT_B_PROTOCOL_REQ message, incoming calls couldn't connect over this B-channel.

Ethernet / LAN

- After receiving a faulty packet the BRICK sometimes stopped receiving any data on certain LANs. This bug has been fixed.

IP

- V.42bis compression sometimes generated conditions that lead to panics in the decompressor. This situation didn't appear often. V42bis compression now works reliably.
- In rare cases, the incorrect IP address was transmitted in the source address field of IP frames. This is fixed.

IPX

- The BRICK now correctly sets the number of ticks (*ipxDestTicks*) for routes more than two hops away. In previous releases this led to frequent dialup connections in certain looped network configurations.
- NETX clients couldn't always find the closest server. This has been fixed. Compared to VLM with Packet Burst and LIP enabled, NETX performs badly over WAN links. Recommendation: Consider upgrading to VLM for clients accessing servers via WAN links.

ISDN

- In certain circumstances the BRICK didn't accept isdnlogin or PPP connections. The source of this problem was found and was corrected in revision 4.3.11.
- Charging information was incorrectly reported in accounting messages (4.3.10 only). This has been corrected.

Setup Tool

- The *ipDenySrcIfIndexMode* field is correctly set to verify / dont_verify when deny entries are added to IP access lists.
- Previously, ip_lapb encapsulation for Leased Line interfaces couldn't be configured under Setup Tool. This has been corrected.

TFTP

- TFTP connections were sometimes lost when large configuration files were loaded via TFTP. The sending of ICMP Source Quench Messages was found to be the cause for this and are no longer sent in response to UDP packets.

X.25

- X.25 Diagnostic and Clear codes are now reported in plain text under Setup Tool's X.25 Monitor.
- Sometimes the system was reported to "hang" when the X.25 link was looped, or the cable wasn't terminated.

4.3 Revision 10: Released: 21.01.97

Features:

Bellcore Conformance Approval

The BRICK now supports the following ISDN protocols:

- National ISDN 1
- National ISDN 2
- Northern Telecom DMS-100
- AT&T 5ESS Custom ISDN

Enhancements:

CAPI

- Performance enhancements in the remote CAPI server.

Bugfixes:

ISDN

- In rare cases, isdnCallState was not adjusted properly and did not reflect the appropriate status.
- Charging information is now correctly reported.

PMX (CM-PRI) only

- The system no longer panics during high incoming-traffic loads on PMX interfaces.
- Under certain conditions in connection with V.110, the BRICK produced unreadable characters. This was found to occur when between 1 and 6 stop bits were received between characters. This has been corrected in Revision 10.

X.25 (RFC 1086 support)

- Under certain conditions, incoming X.25 calls were not accepted on the BRICK. This has been corrected.

Ethernet (CM-BNC/TP)

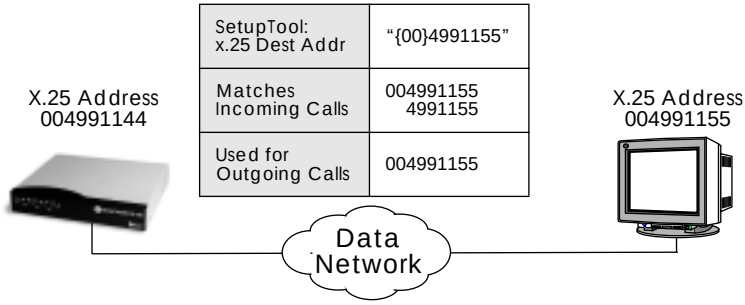
- In rare cases, the BRICK was unable to access the ethernet properly (LAN traffic wasn't received or was received with unusually long delays). The error often appeared on systems that did not answer ping requests, or answered with response times greater than 100ms. Previously, this error could only be corrected with a reboot. Revision 10 fixes this error.

4.3 Revision 9: Released: 13.01.97

Enhancements:

MPX.25 Addressing

The addressing mechanism for MPX25 interfaces has been enhanced. With SetupTool, the remote partner's X.25 address is configured under [X.25][Multiprotocol over X.25][ADD] — "X.25 Destination Address". Previously, the initial 00's were remove from incoming calls and did not allow some calls to connect. The "X.25 Destination Address" field now supports the "{" and "}" characters which can be used to define an optional string of digits to use when matching incoming X.25 calls.



Bugfixes:

PPP Multilink Protocol (MP)

- Incoming calls from some Cisco systems which used In-Band authentication were previously not accepted by the BRICK. This is fixed in Release 4.3, Revision 9.

4.3 Revision 7: Released: 13.12.96

Bugfixes:

PPP

- Using the ShortHold mechanism, the BRICK incorrectly placed inactive PPP interfaces in the “blocked” state instead of the “dormant” state. Only after the interface was reset was it returned to its proper state.

IP

- The ‘netstat -p’ command produced incorrect output and has been corrected in revision 7.

4.3 Revision 6: Released: 09.12.96

Features:

New Communications Module

With revision 6, the BRICK-XM supports the newest version (2.3) of the CM-BNCTP communications module.

Bugfixes:

SetupTool

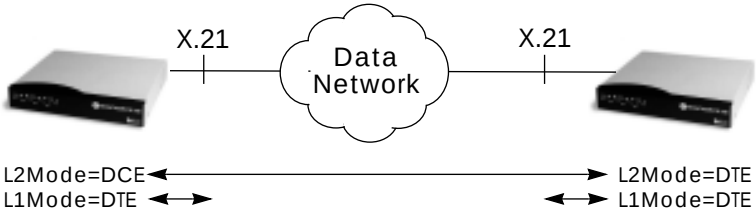
- Monitoring ISDN activity with SetupTools’s ISDN Monitor for extended periods of time sometimes resulted in a system panic. This has been corrected.

4.3 Revision 5: Released: 04.12.96

Features:

X.21 Leased Line Connections

The BRICK-XM supports X.21 leased line connections. The X.21 leased line interface can be a private direct link between two BRICKs with X.21 communications modules, or an official X.21 leased line connection provided by the ISDN provider.

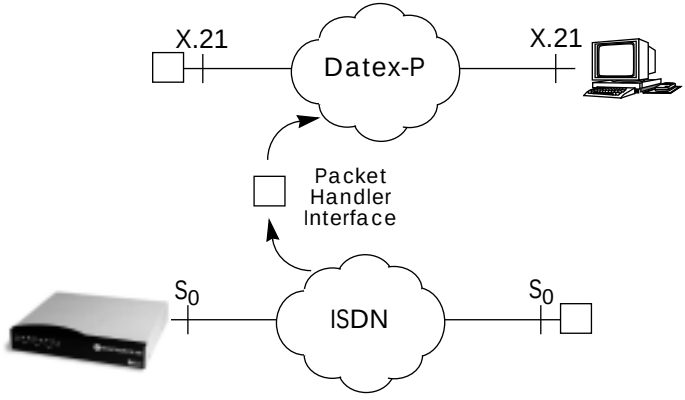


X.21 Leased Line Configuration

Configuring an X.21 leased line connection requires setting the Layer 1 and Layer 2 modes in the ***x21ifTable***. Both sides of the connection must operate as DCE for Layer 1 (*x21IfL1Mode*). At Layer 2 (*x21IfL2Mode*) however, one side must be designated as DTE and one side as DCE.

X.31 in the B-channel for Euro ISDN

The BRICK-XM now supports X.31 in the B-channel according to CaseA and CaseB. This applies only to Euro ISDN and defines alternative procedures that can be used to access public X.25 networks from an S_0 interface.



In both scenarios ISDN hosts access the X.25 network from the S_0 interface through a packet handler interface (PHI) provided by the local ISDN provider. An ISDN call-SETUP is made

using an “Packet Mode” IE-Element. The call is then routed to the PHI automatically.

In CaseA (also known as Minimal Integration) the packet handler must be addressed directly with an ISDN telephone number provided by the network provider. CaseB (also known as Maximal Integration) however, does not require the PHI to be addressed explicitly.

X.31 CaseA Configuration

biboPPPTable:

Set the encapsulation for this interface to “x31_bchan”

biboDialTable

Set the ISDN number for the PHI appropriately.

X.31 CaseB Configuration

biboPPPTable:

Set the encapsulation for this interface to “x31_bchan”

Bugfixes:

PMX (CM-PRI) only

- The BRICK queues data to be sent over ISDN channels in local send-queues. In previous releases, the BRICK didn’t properly clear its send-queues when ISDN channels were closed on which data was currently being sent.

  When ISDN channels were repeatedly closed in this manner the BRICK’s send-queues eventually overflowed and became blocked. Release 4.3 Revision 5 corrects this behaviour.

4.3 Revision 4: Released: 13.11.96

Features:

New X.21 Communications Module

A new version (version 1.1) of the X.21 communications module, CM-X21, is now available. Note however that this version of the X.21 communication module can only be used with system software Release 4.3 Revision 4 or higher.

Continuous Layer 1 State

By default the BRICK-XM sets Layer 1 state to up / down dynamically as X.21 call and indication signals are received via the X.21 interface. With the new X.21 module the BRICK can be configured to assume a constant Layer 1 connection. This is done using the IfLeads field in the *x21IfTable*.

x21IfLeads=disable

This entry "disables" dynamic changing of the Layer 1 state. The BRICK will then assume Layer 1 is always up. By default IfLeads is set to "enable".

Special Note: When configuring a constant Layer 1 state it is very important that the X.21 cabling is attached at BOTH sides of the connection BEFORE booting the system, otherwise various reflection errors may result.

Bugfixes:

IPX

- The configuration of IPX access lists (using ***ipxAllowTable*** and ***ipxDenyTable***) is now working properly. Previously, the BRICK didn't properly filter IPX packets when more than one IPX allow entry was made.

ISDN

- The BRICK-XM now correctly supports the ISDN D-channel AT&T Custom protocol.

UP0 (CM-UP0)

- There was a problem when using the CM-UP0 module in connection with certain ISDN PBXs. This has been corrected.

PMX (CM-PRI)

- The problem with CM-PRI ($1xS_{2M}$) modules and V.110 Support (see *BRICK-XM Last Minute Info*, from 21.10.1996) has been corrected. V.110 bit rate adaptation now works with the CM-PRI module as documented.
- Under software release 4.3 Revision 2, the front panel indicators (LEDs) were not working correctly for slots containing the CM-PRI module. Revision 4 corrects this.

SNMP

- SNMP Link-Down traps are now properly sent, previously only Link-Up traps were sent.

X.25

- The BRICK no longer performs Window and Packet size negotiation during call setup unless specifically required by higher protocol layers or applications.
  In previous releases, the BRICK incorrectly generated WindowSize and PacketSize facilities during call setup for X.31 interfaces that were configured with either *x25LkPrDefWinSize* ≠ 2 or *x25LkPrDefPktSize* ≠ 128.

4.3 Revision 2: Released: 08.11.96

Bugfixes:

CAPI 2.0

- A problem previously reported with outgoing fax polling requests in CAPI 2.0 has been corrected.

ISDN

- The BRICK-XM now correctly supports the ISDN D-channel AT&T Custom protocol.

Miscellaneous

- Memory losses were reported under certain circumstances involving very high system loads. The problem was found in the HDLC module and has been fixed in this release.
- Previously, when using the SetupTool to create an ipDeny entry an ipAllow was incorrectly created instead.
- The internal mechanism for loading configuration files using TFTP has been enhanced. This has been changed mainly for sites with large configuration files; with larger files the TFTP transfer sometimes terminated abruptly with a “tftp timeout” error.

4.3 Revision 1: Released 21.10.96

Features:

Multiprotocol Routing over X.25

The BIANCA/BRICK now supports Multi Protocol Routing (MPR) over X.25, referred to as MPX25 throughout this document. MPX25 allows the BRICK to transmit/receive IP, IPX, Bridging, and AppleTalk packets over X.25 network links by encapsulating these packets into X.25 frame format before transmission.

The BRICK supports MPX25 according to RFC 1356. This RFC is the successor to RFC 877 (which specified how to route IP traffic over X.25) and is upward compatible with 877.

As in ISDN, a point-to-point interface must be configured for each partner you want to reach via MPX25. MPX25 interfaces are defined in the ***x25MprTable***.

x25MprTable

```
brick:>x25mprtable

inx IfIndex(*rw)     Mtu(rw)          Encapsulation(-rw)   NumVC(rw)
    MaxVC(rw)        WinSize(rw)      PktSize(rw)          ShortHold(rw)
    MaxRetries(rw)   BlockTime(rw)    Addr(rw)
```

- *x25MprIfIndex*
  Specifies the interface index of the MPR interface. By convention, the valid range for this parameter is 20001 - 29999. When creating MPX25 interfaces, this parameter may first be set to 0, to allow MPX25 to find the next free value.

- *x25MprMtu* (Default: 1500)
  This parameter is the MTU to be used on the MPX25 interface. It is useful to set the MTU to the same value used on the LAN interface; otherwise the BRICK will be forced to segment data frames.

- *x25MprEncapsulation* (Default: ip_rfc877)
  This parameter specifies the variant of RFC1356 that shall be used for the MPX25 interface, and also specifies the types of traffic, that can be routed over the MPX25 interface. The following settings are available:

  (ip_rfc877) This variant is compatible to RFC877. It uses IP encapsulation and allows only IP-traffic.

  (ip) This variant uses the SNAP encapsulation for IP traffic and is also capable of IP traffic only.

  (ipx) This variant uses the SNAP encapsulation for IPX traffic and is capable of IPX- traffic only.

  (mpr) This variant uses the NULL (Multiplexed) Encapsulation and is capable of IP-, IPX- and BRIDGE-traffic.

  Please note, that for incoming calls other encapsulations are also accepted, as long as they support at least one of the protocols supported by the configured encapsulation.

- *x25MprNumVC* (Default: 1)
  This parameter specifies the number of virtual circuits (VCs) to be used with the MPX25 interface. Please note, that the order of the datagrams cannot be preserved by MPX25 when using multiple VCs. Most LAN protocols assume, that the order of datagrams are preserved by the network. So except for IP, it is strongly recommended, to set this parameter to 1. MPX25 will setup as many VCs as specified by this parameter.

- *x25MprMaxVC* (Default: 1)
  This parameter specifies the maximum number of VCs accepted on this interface. Please note, that the order of the datagrams cannot be preserved by MPX25 when using multiple VCs. Most LAN protocols assume, that the order of datagrams are preserved by the network. So except for IP, it is strongly recommended, to set this parameter to 1. MPX25 will setup as many VCs as specified by this parameter.

- *x25MprWinSize* (Default: 2)
  This parameter specifies the X.25 window size to be used for outgoing X.25 calls.
- *x25MprPktSize* (Default: p128)
  This parameter specifies the X.25 packet size to be used for outgoing X.25 calls.
- *x25MprShortHold* (Default: 60)
  This parameter specifies the inactivity time in seconds after which all connected VCs for the MPX25 interface are cleared. Setting the parameter to 0 disables the Shorthold mechanism and allows VCs to remain open.
- *x25MprMaxRetries* (Default: 5)
  This parameter describes the maximum number of retries, MPX25 will use to setup the first VC for the interface. After that, no further retries are made for the time specified by x25MprBlockTime. The interface changes to the blocked state instead, and rerouting can be applied.
- *x25MprBlockTime* (Default: 300)
  This parameter specifies, how long in seconds the interface will remain in the blocked state.
- *x25MprAddr*
  This parameter specifies the X.25 Address for this interface. It is used for outgoing calls as destination address. Incoming calls with a matching source address are assigned this MPX25 interface.

Configuring MPX25 Interfaces

Requirements

Before an MPX25 interface can be created, the underlying X.25 network must first be configured. At a minimum:

- A valid X.25 license must be installed (***biboLicInfoTable***).
- X.25 routes must be configured (***x25RouteTable***).

Then, once the MPX25 interface is created, it can be used by the higher level protocols (IP, IPX, Bridging) to transport traffic.

Creating an MPX25 Interface

To create a new interface the variables *x25MprIfIndex* and *x25MprAddr* must be set. Initially, the *IfIndex* can be set to 0. The BRICK will automatically adjust this number to the next available value. The *Addr* field must be set to the partner's X.25 address. Note, as defined by the default setting for the *Encapsulation* field, this interface will only support IP traffic.

```
brick:>x25MprTable

inx IfIndex(*rw)      Mtu(rw)          Encapsulation(-rw)    NumVC(rw)
    MaxVC(rw)         WinSize(rw)      PktSize(rw)           ShortHold(rw)
    MaxRetries(rw)    BlockTime(rw)    Addr(rw)

brick:x25MprTable> ifindex=0 addr=499117853321
brick:x25MprTable>
```

You can verify the status of the new interface with the ifstat command.

```
brick:x25MprTable> ifstat

Index  Descr  Type Mtu   Speed St  Ipkts Ies Opkts  Oes  PhysAddress
  [...]
20001  mpx1   x25  1500  0     do  0     0   0      0    point to point
  [...]
brick:x25MprTable>
```

Once the interface is created, it can be used by higher level protocols to transport traffic. For example, to route IP traffic, for the host at 200.1.1.1 over the X.25 MPR interface, an indirect route would be added to the ***ipRouteTable***.

```
brick:x25MprTable> 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)

brick:iproutetable> Dest=200.1.1.1 Ifindex=mpx1 Type=indirect
brick:iproutetable>
```

A complete Example MPX25 Installation is listed at the end of this document.

Extended IP Routing

The BRICK now provides finer control of the routing of IP traffic using an extended IP routing table. Normally, the routing of IP data is dependent solely on the packet's destination address. With the ***ipExtRtTable***, routing decisions can be made based on information contained in the data packet's header such as:

Type of Service (TOS field)
Source IP Address
Source and/or Destination Port

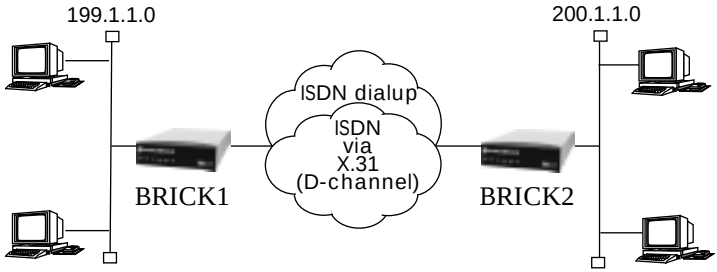
Routing decisions can also be based on;

Source Interface
State of the Destination Interface

When routing IP traffic, the BRICK-XM first consults the ***ipExtRtTable***. Only if a matching route is not found there is the ***ipRouteTable*** consulted. Through the ipExtRtTable, routing decisions can be made based on precise information, therefore careful consideration must be made when defining extended routes.

One of the main advantages of extended IP routing is that IP traffic can be selectively routed over different transport mediums based on your network's needs. Some users require greater bandwidth for bulk data transmissions while others need shorter delays for interactive sessions. Extended IP routing allows installations using multiple technologies (ISDN dialup, leased lines, X.25, and/or X.31 links) to optimize the usage of those technologies.

For example, the two LANs shown on the following page could be connected via an ISDN basic rate interface. For our 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 link.



- Presetting

1. Each BRICK needs to be configured to allow for normal routing via our ISDN dialup link (dialup1, ifindex=10001). See: *Dialup IP over ISDN using PPP* in the User's Guide.

2. Next, we'll need to create an MPX25 (mpx1, ifindex=20001) interface to allow IP traffic to be routed over X.25.
   See Creating an MPX25 Interface.

   - 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***.

     BRICK1:[1]

```
brick1:>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:>Protocol=tcp SrcPort=23 DstAddr=200.1.1.0 DstIfIndex=20001

brick1:>Protocol=tcp DstPort=23 DstAddr=200.1.1.0 DstIfIndex=20001

brick1:>
```

1. The *DstMask* field is generated automatically to a class C network.

BRICK2:[1]

```
brick2:>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)

brick2:>Protocol=tcp SrcPort=23 DstAddr=199.1.1.0 DstIfIndex=20001

brick2:>Protocol=tcp DstPort=23 DstAddr=199.1.1.0 DstIfIndex=20001

brick2:>
```

Two entries for each BRICK are required, so that both incoming and outgoing telnet traffic can be routed via the MPX25 interface.

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.
- Of course multiple extended routes can be defined based on your LAN's requirements. The *Metric1* - *Metric5* objects can be used to prioritize individual routing entries.

1. The *DstMask* field is generated automatically to a class C network.

Extended IP Configuration

Several new features are available on the BRICK that allow for extended IP configuration of IP compatible interfaces. Using the ***ipExtIfTable***, the following features can be individually configured for each interface.

```
brick:>ipExtIfTable

inx  Index(*ro)          RipSend(rw)          RipReceive(rw)
     ProxyArp(rw)        Nat(rw)              NatRmvFin(rw)
     NatTcpTimeout(rw)   NatOtherTimeout(rw)  NatOutXlat(rw)
     Accounting(rw)      TcpSpoofing(rw)
```

RIP Version 2 Support

With the ***ipExtIfTable***, the BRICK now supports both version 1 and version 2 of the Routing Information Protocol (RIP). Using the variables *ipExtIfRipSend* and *ipExtIfRipReceive*, each interface capable of routing IP traffic can be configured individually.

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.

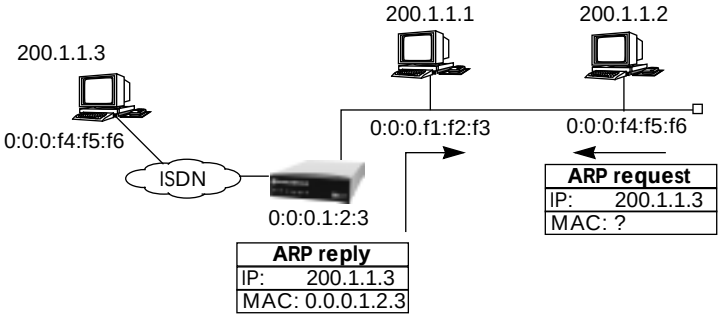
ipExtIfRipReceive can be assigned the values:

ripV1	Accept only RIP V1 packets.
ripV2	Accept only RIP V2 packets.
both	Accept both versions.
none	Ignore RIP packets.

The main difference between Version 1 and Version 2 RIP is that with Version 2, the BRICK also sends the netmask of the propagated IP address. Using version 2 RIP, the BRICK is able to propagate RIP packets to networks that do not use the default netmask for their respective network class.

Proxy ARP

ARP (Address Resolution Protocol) is a technique used to map IP addresses 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.



Each interface in the ***ipExtIfTable*** can be configured for proxy ARP by setting its respective *ipExtIfProxyArp* variable to "on".

When proxy ARP is switched on for this interface, all ARP requests for the target host are answered by the BRICK. IP datagrams destined for this host 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.

Once proxy ARP is enabled, the BRICK interprets all ARP requests it receives. The IP address for which the hardware address is requested, is passed through the ***ipRouteTable***. If the BRICK detects that it must route the request to a different interface from which the request was received on, it answers the request with its own hardware address, i.e, all further packets for this IP address to be sent to the BRICK.

Proxy ARP may cause problems on systems that check for security violations where two IP addresses map to the same physical address.

IP 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. An example session entry might look as follows:

```
brick:>ipExtIfTable

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)

     193.96.238.177   1224              194.45.204.140    80
     6                240               1                 40
     tcp              0 00:07:04.00     0 00:00:27.00     1001(en1)
     2001(Leased)
```

Once a session is terminated, either by disconnection of the TCP session or by timeout, an accounting record is written to the ***biboAdmSyslogTable*** (*Subject*=acct, *Level*=info). Records can also be sent to a loghost using the syslog protocol (see: p. 39).

An accounting record looks like this:

INET:	identifies the subsystem
1.12.96	date, the session was established
13:40:04	time, the session was established
3	session duration from first to last packet
6	protocol identifier 6=tcp, 17=udp 1=icmp
1.2.3.4:4765 / 1000	source ip-address / port / interface
2.3.4.5:21 / 10001	destination ip-address / port / interface
3	packets sent
456	bytes sent
3	packets received
45	bytes receives

Please note, that logging with the syslog protocol is unreliable. In seldom cases accounting records may get lost.

Network Address Translation

Network Address Translation, or NAT allows a router that connects one or more networks to hide a complete LAN behind one IP address.

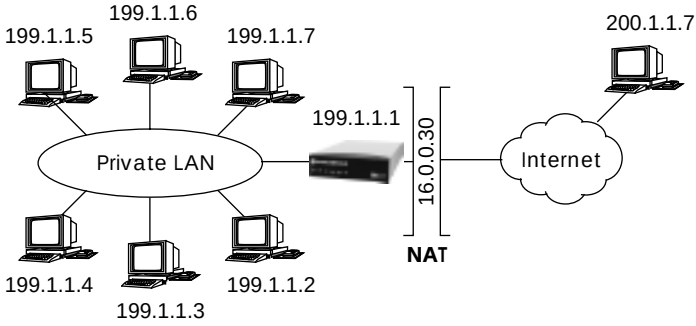
This may be useful in different installations where:

- Security is an issue.

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

The BRICK performs NAT by manipulating all incoming/outgoing IP packets to reflect different source and destination addresses.

In the example installation shown below the BRICK's ISDN interface can be configured for NAT. All hosts on the Private LAN at 199.1.1.0 can still access the external network, but will appear to external hosts as the same host. Connections initiated from outside the private LAN can access local hosts only after local hosts have been explicitly configured to accept connections from external hosts.



Enabling NAT

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:

```
ipRouteIfIndex=10001 ⇨
ipRouteDest=16.0.0.30 ⇨
ipRouteNexthop=16.0.0.30 ⇨
ipRouteMask=255.255.255.255 ⇨
ipRouteType=direct
```

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.

2. The interface the BRICK should perform NAT for must then be enabled in the ***ipExtIfTable***. After locating the correct interface (in our case ifIndex=10001) in the ***ipExtIfTable***, set the *ipExtIfNat* variable to “on”.

```
ipExtIfNat:‹inx›=on
```

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 access to specific hosts on the LAN from external networks, explicit permission must be configured in the ***ipNatPresetTable***. Each entry in this table defines a specific port on a specific host that can be accessed externally.

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 (ftp, telnet, nntp, etc) to allow. Required only if the Protocol uses ports (TCP and UDP).

To allow FTP access to the host at 199.1.1.5, the following entry would be added to the ***ipNatPresetTable***.

```
ipNatPrIfIndex=10001 ⇨
ipNatPrIntAddr=199.1.1.5 ⇨
```

```
ipNatPrProtocol=tcp ⇨
ipNatPrIntPort=21
```

Note that the *IntAddr* of 0.0.0.0 passed the address unchanged to the internal network and can be used as follows:

```
ipNatPrIfIndex=10001 ⇨
ipNatPrIntAddr=0.0.0.0 ⇨
ipNatPrProtocol=icmp
```

Here, all ICMP packets are allowed to enter the private LAN.

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.

```
brick:>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              199.1.1.5          21
    16.0.0.30       456              192.164.19.19      80
    incoming        0 00:01:28.00
01  10001           tcp              199.1.1.2          80
    16.0.0.30       456              136.164.216.11     1605
    outgoing        0 00:18:08.40
02  10001           tcp              199.1.1.3          23
    16.0.0.30       456              122.104.55.10      77
    outgoing        0 00:00:05.00
```

Here we see that three hosts have active sessions.

Entry 00 shows:

The host at 192.164.19.19 has been connected to the FTP service (port 21) on 199.1.1.5.

Entry 01 shows:

Our local host at 199.1.1.2 has an HTTP connection (port 80) open with the host at 136.164.216.11.

Entry 02 shows:

Our local host at 199.1.1.3 currently has a telnet session (port 23) opened with the host at 122.104.55.10.

The Age field specifies the period of time since the last packet was sent received for this session.

Proxies

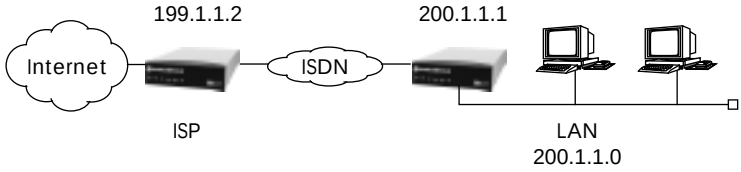
In some cases NAT will not work when port numbers and / or IP addresses are transmitted in the data part of a TCP or UDP session. This is the case for some of the standard Internet Protocols (IP). To allow these protocols to work with NAT, so-called "proxies" have been implemented within the NAT software.

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
- realaudio
- rlogin
- rcp
- rsh

Dynamic IP Address Assignment

The BRICK supports dynamic IP address assignment. The BRICK can be configured as a client, for accepting an IP address from an Internet Service Provider (ISP) for example, or as a server for assigning IP addresses from a pool of available addresses.



Server Mode

The BRICK can act as a server that dynamically assigns IP addresses at connection time. A pool of available IP addresses must first be defined in the ***biboPPPIpAssignTable***. Upon accepting a connection from a client, the BRICK assigns an address from its pool, and automatically creates a host route to the calling client. Once the connection is closed, the BRICK automatically removes the appropriate route

- Configuring the Server

1. First, create a pool of available IP addresses in the ***biboPPPIpAssignTable***. The number of entries you create here will define how many connections that can be established in parallel at any given time. Initially, the *State* field is set to unused but will change dynamically as addresses become used or available.

2. Create (or edit) an entry in the ***biboPPPTable*** for each host the BRICK will assign an IP address to. The *IpAddress* field must be set to “dynamic_server”. If the host is to be identified with PAP or CHAP, set the *Authentication* field appropriately.

3. Each host that is to receive its IP address dynamically, needs to be properly identified at connection time. This can be achieved in either of two ways.

   Creating an entry for each host in the ***biboPPPDialTable***. In this way the calling host is identified using the Calling Number reported in ISDN.

   Configuring PAP or CHAP authentication for the host in the ***biboPPPTable***.

Client Mode

If the BRICK is to be assigned an IP address dynamically from another host (an ISP for example), the BRICK must be configured for client mode. This is done as follows:

- Configuring a Client

1. Create (or edit) an entry in the ***biboPPPTable*** for the partner the BRICK will receive its IP address from. The *ipAddress* field is used to specify what type of address the BRICK uses for this partner. By setting this field to “dynamic_client” the BRICK will create a temporary host route for this partner as long as the connection is established.

Configuring the BRICK for client mode will in most cases only be helpful when NAT (network address translation) is configured which would allow all hosts on the BRICK’s LAN to access the internet using one dynamically assigned IP address. See the section on Network Address Translation for more information.

RADIUS Server Support

Remote Authentication Dial In User Service, or RADIUS, is a client/server security system often used by Internet Service Providers to control access to a network. The server, often a UNIX host, holds a RADIUS database consisting of user authentication data. The BRICK can be configured to operate as a RADIUS client that consults the server at connection time.

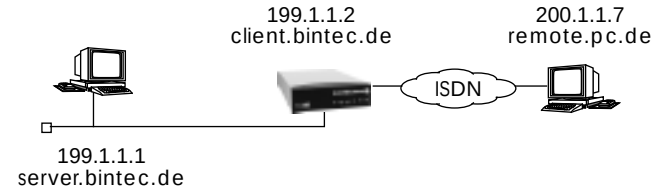
Incoming ISDN calls are first checked for PAP and/or CHAP. If the caller is not identified the BRICK then consults the RADIUS server. If the server authenticates the caller, the BRICK creates a new dialup interface in the ***biboPPPTable*** appropriate to the settings found in the server's /etc/raddb/users file.

Once the connection closes, the BRICK removes all routes referencing this interface and the appropriate partner entry in the ***biboPPPTable***.

- Configuration

1. RADIUS support is enabled on the BRICK by setting the following variables appropriately:

   biboAdmRadiusServer=<*IP Address of server*>
   biboAdmRadiusSecret=<*Client Key*>
   (As defined in the /etc/raddb/clients file.)




```
# /etc/raddb/clients

#Client Name     Key

client.bintec.de  secret
```

```
# /etc/raddb/users

remote  Service-Type=Framed,
        Framed-Protocol=PPP,
        Framed-IP-Address=200.1.1.7
        Framed-IP-Netmask=255.255.255.255,
        User-Password=papsecret
        Chap-Password=chapsecret
        Idle-Timeout=10
```

ISDN Features

V.110 Support

Asynchronous bit rate adaptation is often used in communication with terminal adapters and for connecting to GSM networks from the ISDN. With the appropriate communications hardware, the BIANCA / BRICK-XM is capable of bit rate adaptation according to the V.110 standard. When fitted with the CM-EBRI ($1xS_0$) or the CM-PRI ($1xS_{2M}$) communications modules, the BRICK is fully compatible with the V.110 standard.

- Configuration

  V.110 support can be configured individually for each partner in the ***biboPPPTable***. This is a Layer 1 protocol setting and is set using the *biboPPPLayer1Protocol* variable.

data_64k	HDLC 64Kbit (default setting: used for PPP and LAPB)
data_56k	HDLC 56kBit (reduced bit rate; may be required for USA-Europe connections)
v110_1200	V.110 bit rate adaptation (asynchronous 1200 baud, 8,N,1)[1]

1. Supported by CM-EBRI and CM-PRI. 3. Supported by all modules.

v110_2400	V.110 bit rate adaptation (asynchronous 2400 baud, 8,N,1)[1]
v110_4800	V.110 bit rate adaptation (asynchronous 4800 baud, 8,N,1)[1]
v110_9600	V.110 bit rate adaptation (asynchronous 9600 baud, 8,N,1)[1]
v110_14400	V.110 bit rate adaptation (asynchronous 14400 baud, 8,N,1)[1]
v110_19200	V.110 bit rate adaptation (asynchronous 19200 baud, 8,N,1)[1]
v110_38400	V.110 bit rate adaptation (asynchronous 38400 baud, 8,N,1)[1]
modem	Modem V.22bis (2400 baud), 8,N,1)[1]
dovb	Data 56kBit over Voice Bearer Signalling[3]

For outgoing ISDN calls the Layer1Protocol setting is signalled via the D channel.

- Reception of Incoming Calls

  For ISDN partners that can be identified by the ISDN Calling Party's Number, the *Layer1Protocol* settings will be adjusted appropriately.

  If the caller can not be identified in this way, identification must be performed "inband" using PPP. The setting of the Layer1Protocol for incoming calls can also be performed based on D channel signalling elements. This can be achieved by setting *isdnDspItem* to "ppp".

  In cases where the D channel signalling elements are incorrectly transmitted (through some switching stations and PABXs) another option is available for configuring the layer 1 protocol for incoming calls. This can be configured in the ***isdnDispatchTable*** by setting *isdnDspItem* to one of the following values.

  ppp_v110_1200
  ppp_v110_2400
  ppp_v110_4800
  ppp_v110_9600

1. Supported by CM-EBRI only.

ppp_v110_14400
ppp_v110_19200
ppp_v110_38400

If this mechanism is used, a separate local number (MSN) must be reserved for this *DispatchItem* since the *Layer1Protocol* settings are adjusted to the incoming call's settings.

In most cases, incoming calls can be configured using DispatchItem=ppp since the layer 1 protocol settings are adjusted automatically.

56 kbit Support

The BRICK can now be used with ISDN interfaces that are limited to a bandwidth of 56 kbits. Each ISDN partner must be configured in the ***biboPPPTable*** by setting *Layer1Protocol* to data_56k. The 56kbit mode can then identified by the receiving station via signalling elements in the D channel.

56 kbit mode is supported by all ISDN communications modules (CM-1EBRI, CM-1BRI, CM-2BRI, CM-2UP0, CM-PRI) and is often required for ISDN connections to and from the USA.

Data over Voice Bearer (DOVB) Support for IP Routing

In the USA, ISDN lines that use speech signalling are often provided free of charge for calls within the local calling area; data connections within the local area however are chargeable.

Outgoing Calls

With DOVB it is possible to configure outgoing calls so that charging for data connections within the local area is avoided. The idea is to establish data calls by using voice signalling to avoid charging. This feature is configured in the ***biboPPPTable*** by setting *Layer1Protocol* to “dovb”.

Incoming Calls

Incoming DOVB ISDN calls cannot be identified automatically by the BRICK. Here, a separate entry must be made in the ***isdnDispatchTable***. For example:

```
StkNumber=0 DspItem=ppp LocalNumber=4711
StkNumber=0 DspItem=dovb LocalNumber=4711
```

The DspItem entry for “dovb” can be made for the same local number as the “ppp” entry. The result is that incoming Voice Bearer calls are not accepted per modem, but from the DOVB entry.

In such cases, the DOVB entry overwrites the PPP entry for incoming voice calls.

V.110 Support for ISDNlogin

Through D-channel signalling, isdnlogin can now accept incoming calls with V.110. Connections to V.110 stations can also be established with isdnlogin when the appropriate layer 1 protocol is supplied on the command line, for example:

```
isdnlogin 123 v110_9600
```

The following layer 1 protocols can be used with isdnlogin command.

v110_1200	v110_2400	v110_4800
v110_9600	v110_14400	v110_19200
v110_38400	modem	dovb
56k		

Miscellaneous Features

Sending Syslog Message to Remote Hosts

The BRICK can be configured to send syslog messages to remote hosts on the network using the ***biboAdmLogHostTable***.

The mechanism for sending syslog messages to remote hosts, however, has been simplified since the user documentation was published. Note that for PCs, *DIME Syslog*, included in *DIME Tools*, can also be used. See the documentation for *BRICKware for Windows* for instruction.

The ***biboAdmLogHostTable*** contains four fields as follows:

biboAdmLogHostTable				
OID	Column Name	Type	Index	R/W
.1	Addr	IP	*	RW
.2	Level (—)	E		RW
.4	Facility	E		RW
.5	Type	E		RW

• *biboAdmLogHostAddr*

IP address where syslog messages are sent.

• *biboAdmLogHostLevel*

Priorities lower or equal this level are sent to loghost. The delete priority is used to discard the whole entry.

Valid range:

emerg(1)	alert(2)
crit(3)	err(4)
warning(5)	notice(6)
info(7)	debug(8)
delete(9)	

Default value: emerg

• *biboAdmLogHostFacility*

The facility that initiated the message.

Valid range:

local0(1)	local1(2)

loca2(3)	local3(4)
local4(5)	local5(6)
local6(7)	local7(8)

• *biboAdmLogHostType*

The subject of the messages that are being generated. If set to "system" all subjects except accounting (acct(2)) messages are sent, "all" means all subjects are sent.

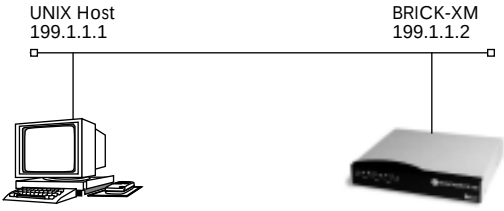
Valid range:

system(1)	acct(2)
all(3)	

Example Syslog Setup

The example shown below would allow syslog messages from all of the BRICK's subsystems (except accounting) with a priority level ≤ warning to be sent to the host at 199.1.1.1.

The remote host's syslog daemon must be configured to accept syslog messages from the local5 facility. For the proper configuration of your syslog daemon, you may need to refer to your local man pages.




```
#      /etc/syslog.conf
# syslog configuration file.
#
*.local5   /var/log/brick.log
```

biboAdmLogHostTable	
Addr	199.1.1.1
Level	warning
Facility	local5
Type	system

Figure 1: Sample LogHostTable

Configuring a Hyperchannel for Leased Lines

Using the ***pmxIfTable***, a hyperchannel can also be configured for leased lines with two BRICKs with CM-PRI S_{2M} modules (a so called back-to-back installation) as follows:

1. The ***pmxIfTable*** is displayed as follows:

```
brick:>pmxIfTable

    inx Index(*ro)          Selftest(ro)          Layer1State(rw)
    Layer1Mode(ro)          Layer1Framing(rw)     Layer1LineCode(rw)
    ChannelMode(rw)         LoopbackMode(rw)      Errors(ro)

00  3000                    successful            no_signal
    e1                      e1_crc                e1_hdb3
    e1_standard             no_loop               0

brick:pmxIfTable>
```

2. To set up the channel's mode for 1 D channel at 64 kbit / sec and 1 B channel at 1920 kbit / sec, enter the following after displaying the ***pmxIfTable***:

   ChannelMode=e1_H12

3. Turn ISDN auto configuration off with:

   isdnIfAutoconfig=off

4. Then delete the ISDN stack in the ***isdnStkTable*** with:

   ProtocolProfile=not_used

5. In the ***isndChTable*** create a new leased line interface.This entry also creates a new entry in the ***ifTable***

   `Type:1=leased_dte`

6. The hyperchannel is configured. All that's left to do now is create the appropriate routes (in the X.25, IP, or IPX route tables) to allow traffic to use the new hyperchannel.

Access with X.25

The BRICK-XM can *receive* X.25 calls even if no X.25 license is present.

This means you can now access your BRICK-XM using X.25—over all possible media, i.e. via a dedicated X.21 interface module, using X.31 on a D-channel, via Ethernet, or via Token Ring—for remote installation of an X.25 license and subsequent configuration.

XON/XOFF protocol now supported

The serial console interface also supports the XON/XOFF protocol, to allow slow terminals to reduce the output speed of the BRICK.

Switchable V.42bis Data Compression in CAPI 2.0

V.42bis data compression can also be switched off on a per call basis by appending an »N« or »n« to the called party number.

New Applications

Debug Application

A new **debug** command is now available from the SNMP shell.The debug command can be used to selectively display debugging information originating from the BRICK's various subsystems.

The syntax is as follows:

debug [**show** | **all** | [<*subs*> [<*subs*> ...]]]

The debug command followed by "show" lists all possible subsystems that can be debugged. If "all" is used all messages are displayed to the screen. Also, one or more subsystem names can be used to display debug messages from different subsystems.

Capitrace Application

The capitrace program, now included with the latest version of *BRICK Tools for UNIX*, enables tracing and interpretation of CAPI messages and displays all CAPI messages sent and received by the BRICK. The environment variable CAPI_HOST must be set to the IP address of the BRICK to trace CAPI messages on.

The capitrace program is used as follows:

capitrace [**-hsl**]

-h hexadecimal output (default)
: Print a hexdump of the entire CAPI message. This option is activated by default (if no options are specified).

-s short output
: Only print at the end of the information line the application ID and a connection identifier in the form "(application/identifier)" and the name of the CAPI message.

-l long output (default)
: Give a detailed interpretation of each

parameter included in the CAPI message.
This option is activated by default.

Each message displayed is preceded by a line containing the following information:

- A time stamp in the format “seconds.milliseconds” that reflects the local time at the BRICK when it sent or received that message.
- A flag signalling whether the message was sent “X” or received “R” by the BRICK.
- A sequence count that is incremented by one for every message. Note that trace messages may be dropped due to overrun conditions. This can be detected by missing sequence numbers.
- The length in bytes of the message in the format “CAPI[bytes]”.

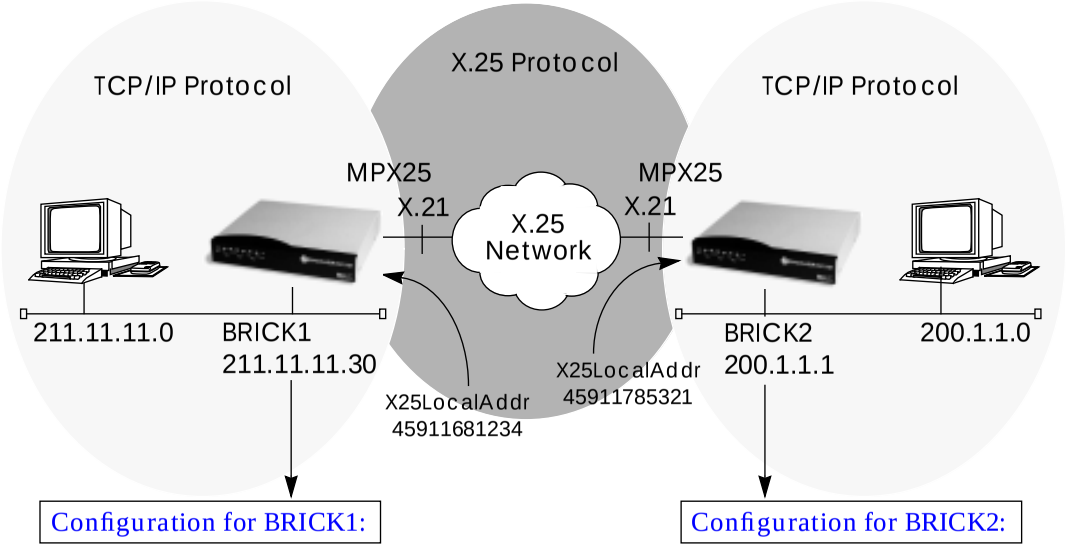
Enhancement in Release 4.3.11

- Names used in the CAPI Message parameters (when long output is generated) now adhere to the CAPI 1.1 and 2.0 standards. Each message displayed is preceded by a line containing the following information:

  Timestamp (“seconds.miliseconds” in localtime)
  Sent / Received Flag (‘X’ = sent, ‘R’ = received)
  CAPI-Message-Name (ASCII string)
  CAPI-Message-Command (0xABXY)
  (AB = <*subcommand*> XY = <*command*>)
  Tracer-Message-Number (#<*decimal*>)
  CAPI-Message-Length (len=<*decimal*>)
  Application-ID (appl=<*decimal*>)
  CAPI-Message-Number
  (messno=0x<*hexadecimal*>)
  Connection-Identifier
  (ident=0x<*hexadecimal*> (short output only))

Bugfixes in Release 4.3.11

- The capitrace application occasionally printed messages twice; this bug has been fixed.
- A minor change has been implemented in the capitrace application to avoid segmentation faults / core dumps on UNIX machines when decoding incorrect CAPI messages.



Example MPX25 Installation

Configuration for BRICK1:

```
> x21IfTable

inx Index(*ro)          L1State(ro)         L1Mode(rw)
    IfLeads(rw)         Speed(rw)           L2Mode(rw)
    SpeedReal(ro)       RxPackets(ro)       RxOctets(ro)
    TxPackets(ro)       TxOctets(ro)        RxResets(ro)
    RxAborts(ro)        RxOverruns(ro)      RxCRCErrors(ro)
    RxGiantFrames(ro)   TxResets(ro)        TxAborts(ro)
    TxUnderruns(ro)     TxGiantFrames(ro)

 00 2000                up                  dte
    enabled             s2048k              auto
    2048000             18210               887090
    18339               898340              2
    0                   0                   0
    1                   0                   0
    0                   0


> x25

x25LocalPadCall( rw):   accept
x25LocalAddr( rw):      "49911681234"


> x25LinkPresetTable

inx IfIndex(*rw)        Addr(rw)            Mode(-rw)
    Modulo(rw)          LIC(rw)             HIC(rw)
    LTC(rw)             HTC(rw)             LOC(rw)
    HOC(rw)             DefPktSize(rw)      DefWinSize(rw)
    MaxPktSize(rw)      MaxWinSize(rw)      L2WinSize(rw)
    L2RetrTimer(rw)     L2RetrCounter(rw)   L2SupervTimer(rw)
    L2IdleTimer(rw)

 00 2000                                    dte
    mod8                0                   0
    1                   2                   0
    0                   p128                2
    p128                2                   2
    1000                10                  10000
    1
```

```
> x25RouteTable

inx SrcIfIndex(*rw)          SrcLinkAddr(rw)          DstIfIndex(*rw)
    DstLinkAddr(rw)          DstLinkAddrMode(-rw)     SrcAddr(rw)
    SrcNSAP(rw)              DstAddr(rw)              DstNSAP(rw)
    ProtocolId(rw)           CallUserData(rw)         RPOA(rw)
    NUI(rw)                  RewritingRule(rw)        Metric(rw)

 00 2000                                              1
    default

    -1                                                -1
                             0                        0

 01 1                                                 2000
    default

    -1                                                -1
                             0                        0

> x25MprTable

inx IfIndex(*rw)             Mtu(rw)                  Encapsulation(-rw)
    NumVC(rw)                MaxVC(rw)                WinSize(rw)
    PktSize(rw)              ShortHold(rw)            MaxRetries(rw)
    BlockTime(rw)            Addr(rw)

 00 20001                    1500                     ip_rfc877
    1                        1                        7
    p128                     5                        5
    5                        "49911785321"

> 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)

 00 211.11.11.0        1000             0                -1
    -1                 -1               211.11.11.30     direct
    netmgmt            3647             255.255.255.0    -1
    .0.0

 01 200.1.1.0          20001            1                -1
    -1                 -1               0.0.0.0          indirect
    local              565              255.255.255.0    -1
    .0.0
```

Configuration for BRICK2:

```
> x21IfTable

inx Index(*ro)          L1State(ro)          L1Mode(rw)
    IfLeads(rw)         Speed(rw)            L2Mode(rw)
    SpeedReal(ro)       RxPackets(ro)        RxOctets(ro)
    TxPackets(ro)       TxOctets(ro)         RxResets(ro)
    RxAborts(ro)        RxOverruns(ro)       RxCRCErrors(ro)
    RxGiantFrames(ro)   TxResets(ro)         TxAborts(ro)
    TxUnderruns(ro)     TxGiantFrames(ro)

 00 2000                up                   dte
    enabled             s2048k               auto
    2048000             19051                908931
    19156               898484               471
    8661                2                    67721
    74                  0                    0
    0                   0


> x25

x25LocalPadCall( rw):   accept
x25LocalAddr( rw):      "49911785321"


> x25LinkPresetTable

inx IfIndex(*rw)        Addr(rw)             Mode(-rw)
    Modulo(rw)          LIC(rw)              HIC(rw)
    LTC(rw)             HTC(rw)              LOC(rw)
    HOC(rw)             DefPktSize(rw)       DefWinSize(rw)
    MaxPktSize(rw)      MaxWinSize(rw)       L2WinSize(rw)
    L2RetrTimer(rw)     L2RetrCounter(rw)    L2SupervTimer(rw)
    L2IdleTimer(rw)

 00 2000                                     dte
    mod8                0                    0
    1                   2                    0
    0                   p128                 2
    p128                2                    2
    1000                10                   10000
    1
```

```
> x25RouteTable

inx SrcIfIndex(*rw)          SrcLinkAddr(rw)          DstIfIndex(*rw)
    DstLinkAddr(rw)          DstLinkAddrMode(-rw)     SrcAddr(rw)
    SrcNSAP(rw)              DstAddr(rw)              DstNSAP(rw)
    ProtocolId(rw)           CallUserData(rw)         RPOA(rw)
    NUI(rw)                  RewritingRule(rw)        Metric(rw)

 00 2000                                              1
                             default

    -1                                                -1
                             0                        0

 01 1                                                 2000
                             default

    -1                                                -1
                             0                        0

> x25MprTable

inx IfIndex(*rw)             Mtu(rw)                  Encapsulation(-rw)
    NumVC(rw)                MaxVC(rw)                WinSize(rw)
    PktSize(rw)              ShortHold(rw)            MaxRetries(rw)
    BlockTime(rw)            Addr(rw)

 00 20001                    1500                     ip_rfc877
    1                        1                        7
    p128                     5                        5
    5                        "49911681234"

> 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)

 00 200.1.1.0          1000              0                 -1
    -1                 -1                200.1.1.1         direct
    netmgmt            4624              255.255.255.0     -1
    .0.0

 01 211.11.11.0        20001             1                 -1
    -1                 -1                0.0.0.0           indirect
    local              1697              255.255.255.0     -1
    .0.0
```