002d9ce8-0

COLLABORATORS

	TITLE : 002d9ce8-0	

ACTION	*NAME*	*DATE*	*SIGNATURE*
WRITTEN BY		August 7, 2022	

REVISION HISTORY

NUMBER	DATE	DESCRIPTION	NAME

Contents

Chapter 1

002d9ce8-0

1.1 AMIPX Manual

```
                                        AMIPX v1.23

          IPX compatibility library
             by G.J. Peltenburg


~Disclaimer~~~
 Who's involved and who's not.

 Introduction
 What is it?

~Requirements~
 What you need.


~Installation~
 It's so easy...

 Preferences
 For the end user/games player.


 When to use it
 to write networked apps.

 API
 For developers.


 Past
 In case you care...

 Present
 Current functionality.

~Future      ~
 Planned improvements.
```

~Credits~ ~~

1.2 Disclaimer

Disclaimer.

AMIPX is freeware; it may be freely distributed for the greater glory of the Amiga, as long as no files are altered.
Freeware in the case of AMIPX, also includes the right to develop software that uses it - just in case you were not sure of that.

AMIPX is not in any way related to Novell, neither in support, nor in source code; this means that any support requests or hate mail involving AMIPX should be addressed to the author, G.J. Peltenburg, and NOT to Novell.

We make no warranties, either expressed or implied, with respect to the software described in this document, its quality, performance, or fitness for any particular purpose. Any risk concerning it's quality or performance is solely the user's. Should the program prove defective, the user assumes the entire cost of all necessary servicing, repair, or correction and any incidental or consequential damages. In no event will we be liable for direct, indirect or consequential damages resulting from any defect in the software.

1.3 Introduction

Introduction.

AMIPX is designed to make porting PC games to the Amiga a little easier, and thus make it a bit more interesting to publish games for the Amiga again.

To achieve this, AMIPX uses a Programming Interface that is very similar to the PC's IPX API, and uses the same structures to exchange information between the application and the library. Some differences remain, though.

1.4 Requirements

AMIPX requires AmigaOS3.0. It may run on 2.04, but that has not been tested.

In addition, it requires a Sana-II network driver. It no longer needs, nor uses packet filtering.

Also, AMIPX currently only supports Ethernet frame types, and therefore it does NOT yet work with Arcnet hardware, or point-to-point Amiga to PC networks.

amipx.library will not work without a valid amipx.prefs file in ENV:. That

file can be created with the prefs editor included in this distribution.

  AMIPX should run on all current 68k processors, although the use of non-aligned WORDs may rule out the use of the 68000.

  No PPC version is planned yet.

Memory usage

Memory usage has been measured by running keepopen and writing down avaliable memory figures from WB.

One LAN interface, with routing enabled:

LAN	Memory used	
Ariadne.device	85k	
Liana.device	138k	(MTU set to 8192)

One LAN interface, routing disabled:

LAN	Memory used	
Ariadne.device	69.5k	
Liana.device	95.6k	(same MTU as before)

Two LAN interfaces (routing always enabled)

LAN	Memory used	
Ariadne.device & Liana.device	160k	(Liana: same MTU as before)

AMIPX coexists with...

AmiTCP/IP - even on the same network adapter. Should be the case with Envoy and any TCP/IP stack, since the packet types used by IPX differ from those of TCP/IP.

Tested Sana-II device drivers...

Ethernet

Ariadne.device v1.44 - works fine (my own Ethernet card)
A2065.device - works (with actual a2065 card). Does not seem to work with a ConneXion card.

Parallel point-to-point

Liana.device v1.4 - works but performance depends on processor. Set frame type to Ethernet-II and choose two different node addresses.
plip.device - should work. Set frame type to Ethernet-II and choose two different node addresses. Performance should be about equal to that of Liana.device .

```
parnet.device          - Does NOT work: it is not a Sana-II driver but a
                         protocol stack with its own parallel support.


Serial point-to-point
---------------------
slip.device            - may work between two Amigas. NOT compatible with
                         any other type of system.
ppp.device             - may work between two Amigas. NOT compatible with
                         any other type of system.
amippp.device          - ditto
```

I'd like to extend the list, but I need your comments for that. I'm especially interested in Ethernet driver compatibility - after all, AMIPX was designed for Amiga to PC networking.

1.5 Installing AMIPX

Told you it was easy:

Step 1: copy amipx.library to your LIBS: drawer.

Step 2: copy amipx_prefs to your Preferences drawer.

Step 3: use amipx_prefs to set up AMIPX for your network.

You may wish to install AMIPX in more than one bootable partition on your hard drive. If you do, be sure to copy the file amipx.prefs file from ENV: as well, or to set up AMIPX using the preferences editor after booting from that other partition.

Additionally, if you have a router (i.e. if you're a company or filthy rich, or if you want to use one amiga to act as a router for another one), you may want to put the 'keepopen' program in your startup-sequence. This way, your programs don't have to wait as long until AMIPX knows which other networks can be reached.

1.6 Preferences

```
Basics
------
```

AMIPX requires you to set it up with specific information on how to access the network card, and information on your network, before it can do its stuff.

```
Fields
------
```

Interfaces tab

The list on the left can be used to select a network interface. The number in the list corresponds to the network number. By clicking on Add, a new network is inserted before the selected network. To add a network to the end of the list, simply select <end> and then click on Add.
Click on Del to remove a network from the list.

The internetwork-address of the first network in the list is reported to IPX applications.

For every LAN interface you want to use with AMIPX, you will have to enter the following fields:

Network number:
Should be set to the IPX network number of your LAN - which is usually 0 in a LAN consisting of a single segment.

Node:
Used only for network drivers that do not have a default network address. It must be unique within the segment you are connected to.
Ethernet network cards have their own default addresses, so you will not need to set this up for those.

Device:
The location and name of the Sana-II network driver to use. Use the '?' button to pick the driver.

Unit:
The unit number of the network interface to be used with the Sana-II driver - 0 if only one network interface of its type is installed in your computer.

Frame Type:
This determines the makeup of the data part of the packets, and must be set to the same type as the other computers are using. For non-ethernet networks, Ethernet-II MAY work.

Some notes on using multiple LAN interfaces

In order to support the use of the local network address by applications, and just about every application needs it, routing is always enabled if more than one network interface is specified. This is preferable over some kind of virtual network, because applications tend to broadcast for servers, and those broadcasts need to be able to get out onto the real network.

If you use AMIPX with the Liana device, and the other computer is turned off, the computer seems to lock up for several seconds periodically. This is probably the case with other software-only networking solutions as well. To avoid this, you may want to set your primary network to your Ethernet card, if you have one, and disable routing. This way, only the primary network is used by AMIPX.

Notes on network number 0

Network number 0 corresponds to all directly connected networks. Networks accessible only through routers will not be reached, unless the routers are configured to route network number 0.

To successfully play a game like DOOM over such a concatenated network, you need to configure all network segments to network number 0, and set the 'Max. Hops Net 0' parameter to 1 or higher (if three networks are daisy-chained, ther are two hops involved).

Don't do this if you're connected to a corporate network! You might keep the PC's from communicating correctly with the server or with 'real' routers.

You do of course, pay a price when you do this: all packets are copied to all network segments, increasing latency (delay between sending and receiving at the other end), and degrading performance on the Amiga acting as the router.

Routing tab

Disable routing:
This is for those who don't have a router, and only one LAN interface, or those with small memories. It saves some memory, and saves some CPU time by not looking for routers and not routing any packets.
If you disable routing, only the first LAN interface will be used by AMIPX.

Max. Routes:
This is the maximum number of additional networks you want AMIPX to support, besides the directly connected networks. This affects both send performance and memory usage.
If you don't have any routers, you can set this to 0.

Max. Hops Net 0:
This enables concatenating networks transparently. This is the maximum number of routers that packets destined for network 0 (local network) can cross.

Note that setting this above 0 (the default) increases network traffic, and decreases performance. It may be useful if you want to play 3- or 4-player games over 'daisy-chained' links. However, this is not recommended with software-only networking solutions, since these rely heavily on the CPU.

Also note that setting this value higher than 16, has no effect. IPX routing supports just 16 hops.

Search Interval:
This is only effective if routing support is enabled. It determines how frequently the networks are scanned for routers. You'll want this to be long enough so as not to influence overall performance too much, but also short enough to get the routes quickly on startup, although this can be shortened by running the 'keepopen program'. I suggest a value of about 45 seconds.

```
Misc. tab

  Maximum # of sockets

  The socket table is used to get messages to the correct applications. Every
  IPX application uses at least one socket. DOOM uses one socket. Quake may use
  multiple sockets.
  The default is 64 sockets, which should be enough for anything you use at
  home. However, possible future applications (server software) may require this
  to be increased.
  The maximum number of sockets does not influence performance. Performance is
  influenced only by the actual number of sockets.


Saving or Using the modified preferences
----------------------------------------
Since AMIPX is a shared library, it is set up when the first application
opens it. At that time, the Sana-II device drivers are opened, and the
preferences are read. When the last application closes the library, the
devices are closed.
Because of this, depending on the applications running, you may need to
quit them or tell them not to use AMIPX, so that they close the library,
before the new settings take effect.
Don't forget keepopen! Keepopen opens the library, and keeps it open until you
send it a Ctrl-C.

If an IPX application has crashed (and therefore has not closed the library),
rebooting the Amiga is necessary.
```

1.7 When to use it

```
When to use AMIPX.
------------------------------

  AMIPX is best used for porting PC IPX networking code to the Amiga. It is
NOT the protocol of choice for new games. You should use TCP/IP for those,
since that makes it possible to use the Internet as well as a LAN for playing
head-to-head.

  AMIPX does not currently support IPX checksumming, SPX, SAP, or NCP and
is therefore of little use for serious software (such as Netware clients).
```

1.8 AMIPX Applications Programming Interface

```
                AMIPX API


TABLE OF CONTENTS


                AMIPX_GetLocalAddr
```

1.9 AMIPX_GetLocalAddr

```
  NAME
AMIPX_GetLocalAddr -- obtain own network address from IPX driver.

  SYNOPSIS
      AMIPX_GetLocalAddr(addressspace);
          a0

      VOID AMIPX_GetLocalAddr(BYTE addressspace[10]);

  FUNCTION
      This function copies the full internetwork address into the supplied
      array, in transmission order.

  INPUTS
      addressspace  - address buffer of at least 10 bytes.

  RESULTS
      none - it is a simple copy operation

  EXAMPLE

  NOTES
```

1.10 AMIPX_OpenSocket

```
  NAME
AMIPX_OpenSocket -- open an IPX socket.

  SYNOPSIS
      actualsocket = AMIPX_OpenSocket(requestedsocket);
```

```
d0        d0

      WORD AMIPX_OpenSocket(WORD);

  FUNCTION
      This attempts to open a socket with the requested socket number, or
      assigns a free socket number if requestedsocket == 0.

  INPUTS
      requestedsocket - socket number

  RESULTS
      actualsocket will be 0 if the socket table is full, or if the requested
      socket is already in use.

  EXAMPLE
      mysock=AMIPX_OpenSocket(0x869c); // official DOOM socket number
      if(mysock==0)
       panic();                       // socket already in use
```

1.11 AMIPX_CloseSocket

```
  NAME
AMIPX_CloseSocket -- close an IPX socket.

  SYNOPSIS
      AMIPX_CloseSocket(socketnumber);
      d0

VOID AMIPX_CloseSocket(WORD);

  FUNCTION
This closes the socket and cancels any outstanding read requests.

  INPUTS

  RESULTS
none - socket will be closed if open

  EXAMPLE
AMIPX_CloseSocket(mysocknum);

  NOTES
To cancel outstanding Listen ECBs (read requests), you should close
      the socket with this function. Manually changing the ECB's status
      does not work.
```

1.12 AMIPX_SendPacket

```
  NAME
AMIPX_SendPacket -- send an IPX packet.
```

```
  SYNOPSIS
error = AMIPX_SendPacket(ECB);
d0       a0

WORD AMIPX_SendPacket(struct AMIPX_ECB *);

  FUNCTION
This submits a send request to the network card. The call returns
      immediately; by checking the ECB's InUse flag (0 means ready), the
IPX user can see if the packet has been transmitted yet.

  INPITS
ECB - pointer to Event Control Block.

  RESULTS
error - 0 if all went well.

  NOTES
Currently, the only two reasons why AMIPX_SendPacket might fail are:
1) the socket has not been opened.
      2) there is no route to the destination.
```

1.13 AMIPX_ListenForPacket

```
  NAME
AMIPX_ListenForPacket -- Insert ECB in IPX read queue.

  SYNOPSIS
error = AMIPX_ListenForPacket(ECB);
d0            a0

WORD AMIPX_ListenForPacket(struct AMIPX_ECB *);

  FUNCTION
      Inserts ECB in listen queue.
You should initialise the InUse flag of the ECB to 0x1d, then call
      this function.
The InUse flag can be checked to see if a packet has been received.
      InUse will be 0 once a packet has been received.


  INPUTS
ECB - pointer to Event Control Block

  RESULTS
error - 0 if successful.

  EXAMPLES

... // initialise ECB
myECB.InUse=0x1d;
if(AMIPX_ListenForPacket(&myECB)) // insert in queue
 panic(); // socket not open or something
```

```
...
if(myECB.InUse==0) {
 ...      // process packet
       AMIPX_ListenForPacket(&myECB);   // insert again
      }
```

```
  NOTES
You must NEVER change any fields in the ECB while it is in the queue.
This call will fail if the socket has not been opened.
```

1.14 AMIPX_RelinquishControl

```
  NAME
AMIPX_RelinquishControl -- allow for some IPX housekeeping

  SYNOPSIS
AMIPX_RelinquishControl();


VOID AMIPX_RelinquishControl(VOID);

  FUNCTION
This will allow the IPX library to do some clean up.

  INPUTS

  RESULTS

  EXAMPLES
AMIPX_RelinquishControl();

  NOTES
This cleanup is also performed when any other AMIPX function is
called.
```

1.15 AMIPX_GetLocalTarget

```
  NAME
AMIPX_GetLocalTarget -- find network address to send a packet to

  SYNOPSIS
error = AMIPX_GetLocalTarget(internetworkaddress,localtarget);
d0           a0      a1

      WORD AMIPX_GetLocalTarget(BYTE internetwaddr[12],BYTE localtarg[6]);

  FUNCTION
This function attempts to find the node number on the local network,
to which a packet has to be sent, in order to have it end up
at the requested internetworkaddress. This can be used to find the
router/bridge on the local network, that leads to the target network.
The ImmedAddr field of the ECB can then be filled with the local target.
```

```
    INPUTS
  internetworkaddress - an array of 12 bytes, consisting of network
            number (4 bytes), node number (6 bytes) and
            socket number (2 bytes). This is the end address
            where you would want your packet to arrive.

    RESULTS
  error - 0 if successful, nonzero if not found for some reason.

  localtarget     - an array of 6 bytes, filled in with the node
            number of the device on the local network to
            which you should send the packet.

    EXAMPLES

    NOTES
  Currently, not really operational, but succeeds if target network
        number is equal to that of the amiga, and fails if not.
```

1.16 ECB Description

```
                ECB - Event Control Block

The ECB contains all the information on a Send or Receive request, that is
needed by the IPX library, and it is the ECB, that is the real interface.

    FIELDS

                ESR
                        - Event Service Routine, see below. Make NULL unless you
                    provide an ESR.
  InUse     - will be zero once the ECB has been processed.
  CompletionCode - nonzero if an error occurred while processing.
  Socket    - the number of the socket the request should be queued in.
  IPXWork   - reserved for internal use, subject to change and sensitive.
  DWork     - ditto
  ImmedAddr - Node address to send the packet to, ffffffffffff for
        broadcast.
        Note that this field is automatically filled in by AMIPX
                    since version 1.20 - this is done to facilitate the use of
                    multiple network interfaces.
  FragCount - Number of memory fragments that follow.
  Fragment[n] - record:
      FragData - pointer to buffer
      FragSize - size of buffer

  Note that in a Send, all fragments are concatenated to form the actual
  IPX packet. To keep packet sizes low, from v1.20 AMIPX only transmits
  the amount of bytes specified in the IPX header.

  In a Listen, the fragments are filled starting at the first fragment,
  and when that is full, the second, and so on. The total packet size
        figure is only available in the IPX header, since FragCount and
  the fragment definitions are not modified by ListenForPacket.
```

Also note that the ECB is an internal structure, and you must therefore simply assign all fields without resorting to 'swap' functions.

Why use multiple fragments?

With multiple fragments, sending and receiving packets will be only a fraction slower.
The advantage of multiple fragments is: you can split the header from your own data, by assigning a pointer to a header as fragment 0, and assigning a pointer to your 'user buffer' to fragment 1. You can then access header and user data separately.
You might even make a table of initialised headers, and send the same data to all of those targets, without modifying a lot of data - all you need to do is set up the ECBs.

Initialisation

The following fields in the ECB must be initialised before calling AMIPX_ListenForPacket:

ESR must be set to NULL, or to the address of a function to be called once the packet has been processed.

InUse Should be set to 0x1d on the first call to AMIPX_ListenForPacket. On subsequent calls, this field is kept up to date. However, if you use the same ECB for both listening and sending, you MUST set it to 0x1d again after the send has been processed.

FragCount must be set to the number of memory fragments.

Fragment[n] must contain a pointer to a buffer, and the size of that buffer.

The following fields in the ECB must be initialised before calling AMIPX_SendPacket:

ESR must be set to NULL, or to the address of a function to be called once the packet has been processed.

ImmedAddr must be set to the node address to which the packet is to be transmitted. In a single segment network, this is the node-part of the destination address.
(This is not an AMIPX requirement)

FragCount must be set to the number of memory fragments.

Fragment[n] must contain a pointer to a buffer, and the size of that buffer.

1.17 Event Service Routine

```
Event Service Routine

The ESR is an optional pointer to a procedure (i.e. no return value)
that is called when the ecb has been processed.


  void (*ESR)(BYTE caller,struct AMIPX_ECB *ecb)
                          D0                      A0

    The procedure when called, will have two arguments:
  caller - indicates which software calls it - currently only IPX
               is supported, indicated by a value of 0xff.
  ecb    - the address of the ecb. The s

    NOTES
  Because all ESRs are handled by one task, an ESR must be a simple
        routine, that takes up very little time. Typically, an ESR would be
        something like:


  void ESR(BYTE caller,struct AMIPX_ECB *ecb)
        {
   geta4();
   Signal(MainTask,MainTaskSigmask);
        }

        As of version 1.20, it is allowed to call AMIPX_SendPacket and
        AMIPX_ListenForPacket from an ESR.
```

1.18 IPX Header Description

```
IPX header
The IPX header, which consists of 30 bytes, is filled by the IPX user, and
contains information such as destination address and source address.

    FIELDS
  Checksum      can contain a checksum for corrupt packet detection,
          but does not seem to be used by most software.
                      This should be set to 0xffff.

  Length        holds the total length of the packet, including the
          30 bytes of the header.

  Tc        holds the number of hops (transfers from one network
                      segment to another) it took to get here.

  Type        This is the IPX packet type - don't confuse with frame
          type. The packet type should be set to 4 (packet exchange)
          although PC DOOM sets it to 0 (unknown packet).

  Dst       This is the complete internetwork address (network,node,
          socket) of the destination.
```

```
      network   network number, MSB first, 4 bytes.
      node      network card number, MSB first, 6 bytes.
      socket    socket number, MSB first, 2 bytes.

  Src       This is the complete internetwork address (network,node,
          socket) of the sender.


  Note that all fields in the IPX header should be defined as bytes or
  byte arrays, or that system dependent 'swap' functions should be used
  to assign or read these values. Although IPX uses the same byte ordering
  as the amiga, source code would not be portable to other processors
  if you didn't.

  Also note that, although not required, most games use the same socket
  for the source as the target. The reason is of course, that
        broadcasting separately to obtain the socket number to send to, then
        sending to that socket, is a lot more unnecessary work.


    Initialisation
  Before AMIPX_ListenForPacket, NO FIELDS need to be initialised in the
  IPX header.

  Before AMIPX_SendPacket, all fields should be initialised (see above).
```

1.19 Additional Requirements

```
Additional requirements

OpenLibrary only by processes
The first task calling OpenLibrary for amipx.library, MUST be a DOS or CLI
process. This is because the first opener, also reads the communications
preferences. Additional tasks may call OpenLibrary later, and as long as the
library is open, it will work fine.

First opener and last 'closer' must have free signalbit
The initial communications between the library and the subtasks is done through
Exec Signals. One is needed at opening time and at closing time. Again, if
your task is NOT the first to open the library, and it is NOT the last to close
the library, this is of no concern. Especially beware of not freeing a signal
if you aren't sure if your program will be called repeatedly from one CLI.
```

1.20 Past

```
Late november, 1997:
  I decided to write IPX support for the Amiga, after
failing to find existing software say from Novell.

  Sending and receiving IPX packets is quite simple on an Amiga, all it
requires is a Sana-II device driver, and some knowledge of the way IPX data
is organised. I wrote a program that monitored and sent IPX packets, in less
```

than two weeks, starting from scratch, and without prior experience with Sana-II.

Making the API almost the same as that of the PC, and putting all that into a shared library is a lot more work, and it took me some 6 weeks to complete it.

18th of january,1998: First Aminet release. Actually took another week due to Aminet trouble.

29th of january,1998:
Started work on version 1.1. Purpose: to remove AMIPX' dependency on the packet filtering function, that seems to be lacking in many Sana-II drivers, and to make it easier to make future modifications such as multiple network adapters and internal bridging by AMIPX.

1st of february,1998:
Version 1.1 now sends and receives packets.

2nd of february,1998:
SendTask failed to free its IORequest. Fixed.
Tested with liana.device. Works in this version of amipx.

4th of february,1998:
Added ESR support. Now a program can have AMIPX call a -small- routine to, for example, wake up the main task.

5th of february,1998:
Modified amipx.h, amipx_pragmas.h (SAS C pragmas inserted). Modified this document. (API now indexed as well)

6th of februari,1998:
Uploaded version 1.1 on Aminet. Works with ADoom.

7th of februari,1998:
Changed behaviour of AMIPX_SendPacket: it will now fill in the source address of the packet header, because some PC IPX sources assume that IPX does that.

2nd half of februari, 1st half of march:
Added multi-network card support and routing support. Moved the ESR processing into a separate task; ESRs may now take a bit longer. Updated prefs editor to support multiple network cards.

15th of march,1998:
Release 1.20 on Aminet

17th of march,1998:
v1.20 - found out new functionality was inactive due to last minute changes. Fixed.

24th-31st of march,1998:
Changed behaviour of broadcasts on network 0 - network 0 is no longer an ordinary network number, but stands for all directly connected networks.
Added capability to route broadcasts on network 0 - means a lot more traffic, but may be the only way to get a game like DOOM, to cross into a different network.

```
  Packets received with network number 0, have the network number automatically
  filled in now.
  Amipx now also enforces the rule that a packet cannot pass more than 16
  routers.
  Modified fool_doom2: it no longer busy-waits and can be killed using Ctrl-C.
  Program is more complex though.

April, 1998:
  Reintroduced packet filtering for network drivers that support it. Filtering
  was _required_ in AMIPX 1.0 . Unfortunately ariadne.device seemed to be the
  only device supporting packet filtering. Filtering is now used properly,
  i.e. to reduce overhead by rejecting non-IPX packets.
  Modified routing to route any NETBIOS packet (type 20) to all networks as if
  it were a global broadcast.
  Started research on SPX.
  Modified static socket table to be user-configurable.
  New layout for amipx_prefs. It now uses tabs. Mainly done for NTSC hires
  users. (assuming that any are actually out there)
```

1.21 Present

```
What AMIPX currently does:

 1) It sends and receives IPX packets and distributes the received packets
    to the correct application.

 2) It offers almost the same basic API as PC IPX.

 3) It uses routers to access other network segments, and can act as one.

What it does not do:

 1) Support SPX. Perhaps this should be layered on top of IPX in a separate
    library.

 3) IPX checksumming.

 4) Support other frame types than Ethernet.
```

1.22 Future

```
I'm looking into PPP support (for WAN routing by the Amiga).

Of lesser importance is the reported version number of the library: when
loaded, the correct version number is reported, but the standard 'version'
command reports a version of 1.-1 . This may cause problems with the install
script, if it too cannot read the correct revision number, and if it refuses
to replace a library with the same version. It seems to have something to do
with the size of the file.
```

1.23 Credits

Thanks to:

Máximo Piva for his comments.

ClickBoom for their positive response.

Peter McGavin for his comments, and testing with ADoom and a2065.device.

The late Commodore-Amiga, Inc., for their near-perfect Sana-II standard.

AMIPX was written with Manx Aztec C 5.0, and incorporates its resident library startup code. Only a single assembly instruction was needed (jumping to the user-supplied ESR).

How to Contact the author

Email: gpeltenburg@wxs.nl