rxlibnet_eng_guide

COLLABORATORS

	TITLE : rxlibnet_eng_guide	

ACTION	*NAME*	*DATE*	*SIGNATURE*
WRITTEN BY		May 24, 2025	

REVISION HISTORY

NUMBER	DATE	DESCRIPTION	NAME

Contents

Chapter 1

rxlibnet_eng_guide

1.1 main

```
rxlibnet.library 2.1

1. Warning
2. About this guide
3. Distribution
4. Author
5. Introduction
6. System requirements
7. Installation
8. Terms
9. Bugs
10. Structures
11. Functions
12. Function by type
13. Bibliography
14. Note
```

1.2 warning

```
THIS SOFTWARE AND INFORMATION ARE PROVIDED *AS IS*.
ALL USE IS AT YOUR OWN RISK, AND NO LIABILITY OR
RESPONSIBILITY IS ASSUMED. NO WARRANTY IS MADE,
ECCEPT THAT *THIS CODE IS TOTALLY BACKDOORS FREE*.
```

1.3 guide

```
This guide contains links with inline ARexx macros wich
use rmh.library/OpenURL. They will work if and only if
you have both openurl.library and rmh.ibrary installed.

rmh.library is by me and it is included in this archive.

openURL.library is by Troels Walsted Hansen:
```

"This library was created to make it easier for application programmers to
 include clickable URLs in their applications, about windows, etc. ..."

You can find it on aminet at comm/www/OpenURL20.lha

1.4 distribution

rxlibnet.library is FreeWare.

You are free to detribute it as long as the original archive is kept intact.
Commercial use or its inclusion in other software package is prohibited ↩
   without
prior consens from the Author.

1.5 author

I am Alfonso Ranieri .

My e-mail address is alfier@iol.it .

My home page is at http://users.iol.it/alfier/ .

You can find me on:
- #amigaita ircnet~;
- #amyita   ircnet .

1.6 introduction

This library contains net utility and stack specific functions .

This library uses rxsocket.library API so it needs rxsocket.library
installed.

The environment is macro-private: each macro opens bsdsocket.library
and what else must be private (e.g. miami.library) and stores a list
of "things" that must be freed on exit.

See rxsocket.guide for more infos.

1.7 installation

Run the installation script.

1.8 requirements

The library needs AmigaOS, version >=2, and a TCP/IP stack.

1.9 terms

- stem or stemName: a valid ARexx variable name e.g. var, var.0, var.name, var ↩
  ;

- socket: the named space created by socket(), Dup2Docket(), and so on;

- socketfd: the socket descriptor id as an integer value;

- addr or address: the Internet address, in dotted form.
  An Internet address is a 32 bits unsigned long, rappresented in the "dotted"
  form as "a.b.c.d" "a.b.c" "a.b" "a" or as a symbolic name.
  In this library addresses are passed/returned in dotted form, e.g. resolve()
  return the dotted form of its argument or -1.
  I wanted to use the original bsdsocket.library API, but ARexx integer
  implementation makes difficult to use integer as Internet addresses.

- types of arguments: the types used are:

D	any data	--
N	numeric	/N
S	symbol	/S ARexx valid symbol
V	stemName	/V ARexx valid symbol as S but with length<20

1.10 bugs

None known.

1.11 structures

The main structures passed or returned to/from functions are:

the usergroup.library structures:
- struct group returned by GetGRGID(), GetGRNam()
- struct passwd returned by GetPWNam(), GetPWUID()

- group
  - NAME
  - PASSWD
  - GID
  - MEMBERS.0, ... ,MEMBERS.last (last = MEMBERS.NUM-1)
  - MEMBERS.NUM

- passwd
  - NAME
  - PASSWD

- UID
- GID
- GECOS
- DIR
- SHELL

1.12 functions

crypt
CreateICMP
CreateIP
CreateTCP
CreateUDP
GenesisGetGlobalUser
GenesisGetUser
GenesisGetUserName
GenesisIsOnLine
GenesisReloadUserList
GenesisSetGlobalUser
GetGRGID
GetGRNAM
GetPass
GetPWNAM
GetPWUID
GetSalt
help
MiamiClosePF
MiamiCreatePF
MiamiDisallowDNS
MiamiGetPid
MiamiIFIndexToName
MiamiIFNameToIndex
MiamiIsOnline
MiamiOnOffLine
MiamiPCapCompile
MiamiPCapMatch
MiamiSetSocksConn
MiamiSupportsIPV6
MiamiPFNext
ReadICMP
ReadIP
ReadTCP
ReadUDP
SockatMark

1.13 functions by type

Sockets
SockatMark

Usergroup
crypt

```
GetPass
GetGRGID
GetGRNAM
GetPWNAM
GetPWUID
GetSalt

Low level
CreateICMP
CreateIP
CreateTCP
CreateUDP
ReadICMP
ReadIP
ReadTCP
ReadUDP

Miami
MiamiClosePF
MiamiCreatePF
MiamiDisallowDNS
MiamiGetPid
MiamiIFIndexToName
MiamiIFNameToIndex
MiamiIsOnline
MiamiOnOffLine
MiamiPCapCompile
MiamiPCapMatch
MiamiPFNext
MiamiSetSocksConn
MiamiSupportsIPV6

Genesis
GenesisGetGlobalUser
GenesisGetUser
GenesisGetUserName
GenesisIsOnLine
GenesisReloadUserList
GenesisSetGlobaLuser

Various
help
```

1.14 createicmp

```
CreateICMP

Usage: icmp = CreateICMP(stem)
<stem/V>

Creates and returns an icmp header reading its fields from stem.
The fields are:
- DATA          the data of the icmp packet, needed to compute its checksum
- TYPE          icmp type
- CODE          type/code
```

```
- PPTR
- GWADDR
- ID
- SEQ
- VOID
- PMVOID
- NEXTMTU
- NUMADDRS
- WPA
- LIFETIME
- OTIME
- RTIME
- TTIME
- IP
- RADV
- MASK
```

1.15 createip

```
CreateIP

Usage: ip = CreateIP(stem)
<stem/V>

Creates and returns an ip header reading its fields from stem.
The fields are:
- V         default 4
- HL        default 5
- TOS       default 0
- LEN       deafult 20
- ID
- OFF
- TTL       default IPDEFTTL
- P
- SRC       source ip addr in dotted form
- DST       dest ip addr in dotted form
```

1.16 createtcp

```
CreateTCP

Usage: tcp = CreateTCP(stem)
<stem/V>

Creates and returns a tcp header reading its fields from stem.
The fields are:
- DATA     needed to compute the tcp checksum
- SPORT
- DPORT
- SEQ
- ACK
- OFF
```

```
- FLAGS
- WIN
- URP
- SRC      sorce addr in dotted form, needed to compute tcp checksum
- DST      dest addr in dotted form, needed to compute tcp checksum
```

1.17 createudp

```
CreateUDP

Usage: udp = CreateUDP(stem)
<stem/V>

Creates and returns an udp header reading its fields from stem.
The fields are:
- DATA    needed to compute the udp checksum
- SPORT
- DPORT
- SRC     sorce addr in dotted form, needed to compute tcp checksum
- DST     dest addr in dotted form, needed to compute tcp checksum
```

1.18 crypt

```
crypt

Usage: cpasswd=crypt(passwd,set)
<passwd>,<set>

The crypt function performs password encryption.
The algorithm used for encryption is implementation-dependent.

The first argument is the clear password, the second is a salt,
that can be created via GetSalt.

Refer to usergoup.doc .

Example:
- generate a password from an user/pass:
  passwd=crypt(pass,GetSalt(user))
- expression to verify a user/pass login:
  cript(pass,passwd)==passwd

Returns the the cripted password.
```

1.19 genesisgetglobaluser

```
GenesisGetGlobalUser

GENESIS ONLY FUNCTION.
```

```
Usage: res=GenesisGetGlobalUser(stem)
<stem/V>

Writes in stem the Genesis global user, if any.
The fields set are:
- NAME
- PASSWD
- UID
- GID
- GECOS
- DIR
- SHELL
- FLAGS
- MAXTIME
- TIMESERVER    (not yet supported)

Returns an ARexx boolean.
```

1.20 genesisgetuser

```
GenesisGetUser

GENESIS ONLY FUNCTION.

Usage: res=GenesisGetUser(stem,login,passwd,title,flags)
<stem/V>,[login],[passwd],[title],[flags/N]

Writes in stem an user from Genesis database.
Open an "user request window" if the user must be identified.
The fields set are:
- NAME
- PASSWD
- UID
- GID
- GECOS
- DIR
- SHELL
- FLAGS
- MAXTIME
- TIMESERVER    (not yet supported)

Returns an ARexx boolean.
```

1.21 genesisgetusername

```
GenesisGetUserName

GENESIS ONLY FUNCTION.

Usage: res=GenesisGetUserName(userNumer)
<userNumber/N>
```

```
Returns the name of the user number userNumber if it exists.
```

1.22 genesisisonline

```
GenesisIsOnLine

GENESIS ONLY FUNCTION.

Usage: res=GenesisIsOnLine(flags)
[flags]

Checks or sets (if) Genesis (is) online.
Flags can be one of:
- ASKUSER
- FORCE

Returns an ARexx boolean.
```

1.23 genesisreloaduserlist

```
GenesisReloadUserList

GENESIS ONLY FUNCTION.

Usage: res=GenesisReloadUserList()
-

Tells Genesis to reload the users list.
Should be used after an application modified the user database.

Always returns 1.
```

1.24 genesissetglobaluser

```
GenesisSetGlobalUser

GENESIS ONLY FUNCTION.

Usage: res=GenesisSetGlobalUser(stem,login,passwd,title,flags)
<stem/V>,[login],[passwd],[title],[flags]

Logins a new user and sets stem with the user fields.
The fields set are:
- NAME
- PASSWD
- UID
- GID
- GECOS
- DIR
- SHELL
```

```
- FLAGS
- MAXTIME
- TIMESERVER    (not yet supported)

Flags is one or more of:
- ASKUSER
- FORCE
- STAYOPEN

About STAYOPEN :
Many applications use genesis.library to find out wich user
is currently logged.
E.g. YAM uses Genesis corrent logged user, if any.
If you are using a different tcp/ip stack, you can still use
this Genesis feature with this flags.
It tells rxlibnet to stay opened so that also genesis.library
remains opened and the user logged, till you unlog using this
function with an empty user name.
Of course, all that has sense only for non Genesis user :)

See GenesisLogin example in the examples drawer.
```

1.25 getgrgid

```
GetGRGID

Usage: res=GetGRGID(GID,group)
<GID/N>,<group/V>

Searches the group database for the given group id, stopping
at the first found. Fills "group" with a group structure.

Returns an ARexx boolean.
```

1.26 getgrnam

```
GetGRNam

Usage: res=GetGRNam(name,group)
<name>,<group/V>

Searches the group database for the given group name, stopping
at the first found. Fills "group" with a group structure.

Returns an ARexx boolean.
```

1.27 getpass

```
GetPass

Usage: res=GetPass(prompt)
<prompt>

Displays a prompt and read in a password.

Returns the password read.
```

1.28 getpwnam

```
GetPWNam

Usage: res=GetPWNam(name,pass)
<name>,<pass/V>

Searches the user database for the given name, stopping
at the first found. Fills "pass" with a passwd structure.

Returns an ARexx boolean.
```

1.29 getpwuid

```
GetPWUID

Usage: res=GetPWUID(UID,pass)
<UID/N>,<pass/V>

Searches the user database for the given UID, stopping
at the first found. Fills "pass" with a passwd structure.

Returns an ARexx boolean.
```

1.30 getsalt

```
GetSalt

Usage: salt=GetSalt(user)
<user>

GetSalt creates a text string that is suitable to be passed to
crypt() as a settings string.

Returns the salt string.
```

1.31 help

```
help

Usage: helpString=help(funName)
<funName>

Returns the arguments mask string of rxlibnet.library function "funName".
```

1.32 miamiclosepf

```
MiamiClosePF

MIAMI ONLY FUNCTION

Usage: call MiamiClosePF(pfID)
<pfID/N>

Close a packet filter created with CreatPF .

See MiamiCreatePF MiamiPFNext
```

1.33 miamicreatepf

```
MiamiCreatePF

MIAMI ONLY FUNCTION

Usage: pfID =  MiamiCreatePF(dev,signal,maxPks)
<dev>,<signal/N>,[maxPks/N]

Create a packet filter and return its id .
A packet filter will receive every in-out coming packet on
the interface "dev" .

"signal" is a rmh.library/AllocSignal allocated signal, that
will be set any time a packet is received .

maxPks is the max number of packets to store; it must be an
integer greater than 1 .

Returns the packet id or -1 for failure .

Example: see pf.rexx

See MiamiClosePF MiamiPFNext
```

1.34 miamidisallowdns

```
MiamiDisallowDns
```

```
MIAMI ONLY FUNCTION

Usage: call MiamiDisallowDns(1|0)
[status/N]

Disabled extern DNS lookup. Default for status is 0.

Returns always 1.
```

1.35 miamigetpid

```
MiamiGetPid()

MIAMI ONLY FUNCTION

Usage: pid = MiamiGetPid()
-
Returns Miami's internal process descriptor as packed chars.
This value is needed when you want to manipulates routes
directly (see ip2if.rexx) .

If you need just a Process ID (e.g. like in icmp echo packets)
use pragma("ID") .
```

1.36 miamiifindextoname

```
MiamiIFIndexToName

MIAMI ONLY FUNCTION

usage: ifname = MiamiIFIndexToName(index)
<index/N>

Return an interface name from an interface index .
```

1.37 miamiifnametoindex

```
MiamiIFNameToIndex

MIAMI ONLY FUNCTION

usage: ifindex = MiamiIFNameToIndex(name)
<name>

Return an interface index from an interface name .
```

1.38 miamionoffline

```
MiamiOnOffLine

MIAMI ONLY FUNCTION

usage: call MiamiOnOffLine(interface,1|0)
<interface>,[status/N]

Switch the status of the interface. Default value for status is 0.
The functions doen't wait for the switching to complete and
always returns 1.
```

1.39 miamiisonline

```
MiamiIsOnLine

MIAMI ONLY FUNCTION

Usage: res=MiamiIsOnLine(interface,0|1)
<interface>

Tests if the given interface is online.
Returns an ARexx boolean.
```

1.40 miamipfnext

```
MiamiPFNext

MIAMI ONLY FUNCTION

Usage:  pkt = MiamiPFNext(pfID)
<pfID>

Get next filtered packet .
pfID is the packet filter id .

See MiamiClosePF MiamiCreate
```

1.41 miamipcapcompile

```
MiamiPCapCompile

MIAMI ONLY FUNCTION

Usage:  filter = MiamiPCapCompile(expr,interface,prom)
<expr>,[interface],[prom/]

Compile the pcap expression for the interface specified,
```

```
or the "suitable" one if any, with promiscous set if
specified.

Return a string that can be used with MiamiPCapMatch() or Null()
for failure; the reasone of the failure can be found in "PACAPERR" .
The filter can be freed with DROP .

See MiamiPCapMatch
```

1.42 miamipcapmatch

```
MiamiPCapMatch

MIAMI ONLY FUNCTION

Usage:  res = MiamiPCapMatch(filter,packet)
<filter>,<packet>

Match a filter created with MiamiPCapCompile() with a
packet returned by MiamiPFNext() (or whatelse) .

See MiamiPCapCompile
```

1.43 miamisetsocksconn

```
MiamiSetSocksConn

MIAMI ONLY FUNCTION

Usage: res=MiamiSetSocksConn(remote)
<remote/V>

Sets the destination address for the next SOCKS bind() request.
Returns an ARexx boolean. remote is a sockaddr_in.
```

1.44 miamisupportsipv6

```
MiamiSupportsIPV6

MIAMI ONLY FUNCTION

Usage: res=MiamiSupportsIPV6()
--

Tests if the current version of Miami supports the IPV6 protocol.

Returns an ARexx boolean.
```

1.45 readicmp

```
ReadICMP

MIAMI ONLY FUNCTION

Usage: call ReadICMP(pkt,stem)
<pkt>,<stem/V>

Fills stem with an icmp header read from pkt.
pkt must at least 28 bytes or an error 18 is generated.
The fields set are:
- TYPE
- CODE
- CKSUM

(Yessss too lazi to make a better icmp parser  :)
```

1.46 readip

```
ReadIP

Usage: call ReadIP(pkt,stem)
<pkt>,<stem/V>

Fills stem with an ip header read from pkt.
pkt must at least 20 bytes or an error 18 is generated.
The fields set are:
- V
- HL
- TOS
- LEN
- ID
- OFF
- TTL
- P
- SUM
- SRC in dotted form
- DST in dotted form
```

1.47 readtcp

```
ReadTCP

Usage: call ReadTCP(pkt,stem)
<pkt>,<stem/V>

Fills stem with a tcp header read from pkt.
pkt must at least 20 bytes or an error 18 is generated.
The fields set are:
- SPORT
- DPORT
```

- SEQ
- ACK
- OFF
- X2
- FLAGS
- WIN
- SUM
- URP

1.48 readudp

```
ReadUDP

Usage: call ReadUDP(pkt,stem)
<pkt>,<stem/V>

Fills stem with an udp header read from pkt.
pkt must at least 8 bytes or an error 18 is generated.
The fields set are:
- SPORT
- DPORT
- ULEN
- SUM
```

1.49 sockatmark

```
SockAtMark

Usage: res=SockAtMark(socketfd)
<socketfd/N>

MIAMI ONLY FUNCTION

Tests if the socket is in out of band status.

Returns an ARexx boolean.
```

1.50 bibliography

- Quite all rfc ;

- "Unix Network Programming" - W. Richard Stevens PTR Prentice Hall ;

- libraries autodoc from MiamiSDK, AmiTCPSDK and TermiteTCPSDK .

1.51 note

Pointers to deallocate the local environment in the library base is saved
in the fields pr_ExitCode and pr_ExitData of the Process structure of
the macro. At exit a chain of pr_ExitCode(pr_ExitData) is called.
Details are avaible on request.

Some funtions are avaible only if a peculiar stack is running or installed:
- Miami functions are avaible only if Miami is runnning;
- Miami packets filter functions are avaible only if a registered
  version of Miami is running;
- Genesis functions are avaible only if Genesis is installed;
- user group functions are avaible only if the stack running offers
  the usergroup.library .

When a function is not avaible, the user is informed via a requester
and an ARexx error 15 (function not found) is returned.

Miami-registered-only functions returns error if used with Miami not
regitered.

rxsocket.library/IsLibOn can be used to test the environment.