Network

COLLABORATORS

	TITLE : Network	

ACTION	*NAME*	*DATE*	*SIGNATURE*
WRITTEN BY		July 31, 2024	

REVISION HISTORY

NUMBER	DATE	DESCRIPTION	NAME

Contents

1 Network 1

1.1 Audio . . . 1
1.2 background . . . 1
1.3 closenetworkconnexion . . . 3
1.4 closenetworkserver . . . 3
1.5 createnetworkserver . . . 3
1.6 initnetwork . . . 4
1.7 networkclientid . . . 4
1.8 networkevent . . . 4
1.9 networkserverevent . . . 4
1.10 opennetworkconnexion . . . 5
1.11 receivenetworkdata . . . 5
1.12 receivenetworkfile . . . 5
1.13 receivenetworkstring . . . 6
1.14 sendnetworkdata . . . 6
1.15 sendnetworkfile . . . 6
1.16 sendnetworkstring . . . 7

Chapter 1

Network

1.1 Audio

```
  PureBasic - Network

    Networks are widely spreaded all over the world and allow computers
    to communicate easily. PureBasic supports the official Internet
    protocol to exchange data: TCP/IP. This allows to write applications
    or games using this protocol, using the well known ’client-server’
    model. With these commands, it’s possible to create any kind of
    internet like applications (browser, web server, ftp client...)
    or fast multiplayers games. To use these commands, you need a
    TCP/IP stack, like MIAMI or AmiTCP.


     Priliminary explanations

  Commands summary:

     CloseNetworkConnexion
     CloseNetworkServer
     CreateNetworkServer
     InitNetwork
     NetworkClientID
     NetworkEvent
     NetworkServerEvent
     OpenNetworkConnexion
     ReceiveNetworkData
     ReceiveNetworkFile
     ReceiveNetworkString
     SendNetworkData
     SendNetworkFile
     SendNetworkString

     Network Client Demo
     Network Server Demo
```

1.2 background

General Informations:

This piece of text is a little attempt to explain the basics of the client/server model and the TCP/IP protocol. This doesn’t mean that all informations provided are complete nor 100% accurate.

TCP/IP:

This is a software only transfer protocol developed in the 70’s to send and receive data from any location. The goal was to provide a flexible way to send big files without lot of overhead. In few words, the files are splitted down in many little parts (called ’packets’) and sent one by one. Once it’s on the network, the packet can take any way to reach the destination, and it’s the software’s part to repack all the little parts into one file. Each computer must have an own IP Address which is composed of 4 numbers (each number can take a value from 0 to 255) and a subnet mask (4 numbers too). Ex:

```
Address IP : 192.0.3.25
Subnet mask: 255.255.0.0
```

An IP address must be unique on the network, else there is a conflict (the packets don’t know which computer to go to). On a local network (LAN: Local Area Network) the subnet mask must be the same on all computers, else it will have some problems.

Special IPs:

```
127.0.0.1: Local IP. Each computer has this IP which represents himself.
           (called ’Loopback’ too). This IP is very handy for programmers
           (you can test the client/server programs without being
           connected to any network)

255.255.255.255: never use this one, it’s reserved for Broadcasting.
```

Client/Server:

This is a generic term which is widely used thanks to the internet. You guessed it, internet itself is a client/server like entity. Here is a little graphic to show how it looks:

```
  Computer1
      \
       \
        \
         \
          \
         Server ------ Computer2
           |
           |
           |
        Computer3


  Ok, so if the Computer1 wants to send something to Computer2,
  it must send the data to the server and the server will send
  it to Computer2. The server is a bit like a dispatcher. It
  takes the data from a computer and sends it to another (or
  sends the requester information to this computer ). A server
  can have any number of clients.

  Maybe these little informations help you to build nice and fast
  internet based applications with PureBasic !


        See you,

              AlphaSND.
```

1.3 closenetworkconnexion

```
    SYNTAX
  CloseNetworkConnexion()

    STATEMENT
  Close the current connexion and send a notification to the server.
```

1.4 closenetworkserver

```
    SYNTAX
  CloseNetworkServer()

    STATEMENT
  Shutdown the currently running server. All clients connected to this server
  are automatically removed. The port is freed and can be reused by another
  application.
```

1.5 createnetworkserver

```
    SYNTAX
  Result = CreateNetworkServer(Port)

    FUNCTION
  Create a new network server on the local computer at the
  specified port. Port values can go from 6000 to 7000
  (this is a recommended area space). Any number of servers
  can run simultaneously on the same computer but not with the
  same port number. If the 'Result' is NULL, the server can't
  be created (port in use), else the server has been correctly
```

```
created and is ready to use.


Port: Port number for this server
```

1.6 initnetwork

```
  SYNTAX
Result.l = InitNetwork()

  FUNCTION
This is the initroutine that always must be called before using
any other routines in the Network library. This function attempts
to open the 'bsdsocket.library'. If the 'Result is NULL, there is
no TCP/IP stack available on the system, otherwise everything is
initialized correctly.
```

1.7 networkclientid

```
  SYNTAX
ClientID = NetworkClientID()

  STATEMENT
This command is only needed on the server side. It's needed to know
which client has sent the data.
```

1.8 networkevent

```
  SYNTAX
Result = NetworkEvent()

  STATEMENT
Not NULL if an information has been received via the
Network and needs to be processed. After a NetworkEvent(),
you can typically use commands like: ReceiveNetworkString(),
ReceiveNetworkData(), etc..
```

1.9 networkserverevent

```
  SYNTAX
EventInfo = NetworkServerEvent()

  STATEMENT
Returns not NULL if an information has been received from a
client actually connected to the server. To know which client
has sent something, just use the NetworkClientID() command.
```

```
The return 'EventInfo' can take several values:

0: nothing has happened on the server.
2: a client has sent raw data
3: a client has sent a string (with SendNetworkString())
4: a client has quitted the connexion to the server
5: a client has sent a file (with SendNetworkFile())
```

1.10 opennetworkconnexion

```
  SYNTAX
Result = OpenNetworkConnexion(ServerName$, Port)

  STATEMENT
Try to open a connexion on the specified server. 'ServerName$'
can be an IP address or a full name (ie: "127.0.0.1" or
"ftp.home.net"). If the connexion has been granted by the server
the Result is not NULL, else the connexion has failed.


ServerName$: Name or IP address of the computer which hosts the
             server to connect.

Port: Port number of the running server (see CreateNetworkServer).
```

1.11 receivenetworkdata

```
  SYNTAX
ReceiveNetworkData(ClientID, *DataBuffer, Length)

  STATEMENT
Receive raw data from the specified client. This command can be used
by both client and server applications. On server side, 'ClientID' is
the client which has sent the String. On a client side, just use '0' as
'ClientID' to get the data which is actually in the network queue.

The data is read into the specified *DataBuffer.
```

1.12 receivenetworkfile

```
  SYNTAX
ReceiveNetworkFile(ClientID, FileName$)

  STATEMENT
Receive a file from the specified client. This command can be used
by both client and server applications. On server side, 'ClientID'
is the client which has sent the String. On a client side, just use
'0' as 'ClientID' to get the string which is actually in the network
queue.
```

```
The file must have been sent by using the specific SendNetworkFile()
command.
```

1.13 receivenetworkstring

```
  SYNTAX
String$ = ReceiveNetworkString(ClientID)

  STATEMENT
Receive a string from the specified client. This command can be used
by both client and server applications. On server side, 'ClientID'
is the client which has sent the String. On a client side, just use
'0' as 'ClientID' to get the string which is actually in the network
queue.

The string must have been sent by using the specific SendNetworkString()
command.
```

1.14 sendnetworkdata

```
  SYNTAX
SendNetworkData(ClientID, *MemoryBuffer, Length)

  STATEMENT
Send raw data to the specified client. This command can be used
by both client and server applications. On server side, 'ClientID'
is the client which should receive this data. On a client side,
just use '0' as 'ClientID' to send the data via the current connexion
(created with OpenNetworkConnexion()).
```

1.15 sendnetworkfile

```
  SYNTAX
SendNetworkFile(ClientID, FileName$)

  STATEMENT
Send a full file to the specified client. This command can be used
by both client and server applications. On server side, 'ClientID'
is the client which should receive this data. On a client side, just
use '0' as 'ClientID' to send the data via the current connexion
(created with OpenNetworkConnexion()).

The file is sent using very specific (and proof) methods. It must be
received with the ReceiveNetworkFile() command.

This command locks the program execution until the whole file
has been send.
```

1.16 sendnetworkstring

```
  SYNTAX
SendNetworkFile(ClientID, String$)

  STATEMENT
Send a string to the specified client. This command can be used
by both client and server applications. On server side, 'ClientID'
is the client which should receive this data. On a client side,
just use '0' as 'ClientID' to send the data via the current connexion
(created with OpenNetworkConnexion()).

The string is sent using very specific (and proof) methods. It must be
received with the ReceiveNetworkString() command.
```