devP2P library helpfile

Table of contents

Overview

devP2P is peer-to-peer cross platform library that is used to establish Virtual Private Network between two devP2P instances over internet. All communication between those peers is **encrypted** and **secured**. Peers are able to forward local and remote ports, send messages, transfer files, route complete network, etc.

TCP and **UDP** packets can be used for transport, and several techniques are available to make direct NAT 2 NAT connection between peers. And if direct connection is not possible (in really rare cases), devP2P can be used as relay between other instances as well, or use relays to establish fast connection in 100% of the cases.

AES128/AES192/AES256 encrypts all packets between two peers using password of your choice - unbreakable security is in place! You can choose your own passwords, or can use internal Diffie-Hellman algorithm for password generation.

devP2P can be used to connect through directly through VPN to your SQL server (which is mapped through local port), share pictures, documents and files. You can also use it to provide "remote desktop" feature through the devP2P. Virtually any service that uses TCP or UDP can be "mapped" to work over devP2P.

Platforms

Windows
Mac OSX
Linux
BSD

License Agreement

Secure Plus d.o.o SOFTWARE LICENSE AGREEMENT

This is a legal agreement between you (either an individual or an entity) and Secure Plus d.o.o. ("Secure Plus"). By installing the enclosed software, you are agreeing to be bound by the terms of this Agreement. If you do not agree to the terms of this Agreement, promptly return the software and the accompanying items to the place you obtained them for a full refund. If you need to return the software, you must prepay shipping and either insure the package or assume all risk of loss or damage in transit.

Secure Plus LICENSE
1. GRANT OF LICENSE TO USE. The Secure Plus Software product that accompanies this license is referred to herein as "SOFTWARE." Secure Plus grants to you as an individual, a personal, non-exclusive license to make and use the SOFTWARE for the sole purpose of designing, developing, and testing your software product(s). Secure Plus grants to you the limited right to use only one copy of the Software on a single computer in the manner set forth in this agreement. If you are an entity, Secure Plus grants you the right to designate one individual within your organization to have the right to use the SOFTWARE in the manner provided above. Secure Plus reserves all rights not expressly granted.

2. UPDATES. Upon receipt of future updates of the SOFTWARE (including without limitation the Redistributable Code)(an "UPDATE"), you may use or transfer the UPDATE only in conjunction with your then-existing SOFTWARE. The SOFTWARE and all UPDATES (including bug fixes and error corrections) shall be provided by Secure Plus. To you and are licensed as a single product, and the UPDATES may not be separated from the SOFTWARE for use by more than one user at any time.

3. COPYRIGHT. The SOFTWARE is owned by Secure Plus or its suppliers and is protected by copyright laws and international treaty provisions. Therefore, you must treat the SOFTWARE like any other copyrighted material (e.g., a book or musical recording). You may not use or copy the SOFTWARE or any accompanying written materials for any purposes other than what is described in this Agreement. Secure Plus warrants that Secure Plus is the sole owner of all patents, copyrights or other applicable intellectual property rights in and to the SOFTWARE unless otherwise indicated in the documentation for the SOFTWARE. Secure Plus shall defend, indemnify, and hold Licensee harmless from any third party claims, including reasonable attorneys' fees, alleging that Software (including without limitation Sample Code) licensed hereunder infringes or misappropriates third party intellectual property rights.

4. OTHER RESTRICTIONS. You may not rent or lease the SOFTWARE, but you may transfer the SOFTWARE and accompanying written materials on a permanent basis, provided you retain no copies and the recipient agrees to the terms of this Agreement. You may not reverse-engineer, decompile, or disassemble the SOFTWARE except to the extent such foregoing restriction is expressly prohibited by applicable law.

5. REDISTRIBUTABLE CODE. Portions of the SOFTWARE (specifically the run time modules in binary form) are designated as "Redistributable Code", subject to the Distribution Requirements described below.

6. SAMPLE CODE. Secure Plus grants you the right to use and modify the source code version of the included Sample Code for the sole purpose of designing, developing, testing and supporting your software products. You may also reproduce and distribute the Sample Code in object code form along with any modifications you make to the Sample Code, provided that you comply with the Distribution Requirements described below. For purposes of this section, "modifications" shall mean enhancements to the functionality of the Sample Code.

7. SOURCE CODE. If you have purchased the SOFTWARE source code, you may not re-distribute the source code, nor may you copy it into your own projects. Secure Plus retains the copyright to the SOFTWARE source code. You have no right to change or use source code for 3rd party components or applications. Source code is provided only for your storage and protection. This agreement allows you to obtain access to fix and update the software's source code under special circumstances, such as to provide support to your end user customers to whom you have distributed Redistributable Code in conformance with Section 8 below, or if the Secure Plus goes out of business.

8. DISTRIBUTION REQUIREMENTS. Notwithstanding section 4 above, you are authorized to redistribute the Sample Code and/or Redistributable Code, (collectively "REDISTRIBUTABLE COMPONENTS") as described in Sections 5 and 6, only if you
(a) distribute them in conjunction with and as part of your software product that adds primary and significant functionality to the REDISTRIBUTABLE COMPONENTS ;
(b) do not permit further redistribution of the REDISTRIBUTABLE COMPONENTS by your end-user customers ;
(c) do not use Secure Plus'es name, logo, or trademarks to market your software application product ;
(d) include a valid copyright notice on your software product ; and
(e) agree to indemnify, hold harmless, and defend Secure Plus from and against any third party claims or lawsuits, including reasonable attorney's fees, to the extent arising or resulting from your material breach of your obligations under this agreement.
Secure Plus reserves all rights not expressly granted. The license in this section to distribute REDISTRIBUTABLE COMPONENTS is royalty-free, provided that you do not make any modifications to any of the REDISTRIBUTABLE COMPONENTS. Contact Secure Plus for the applicable royalties due and other licensing terms for all other uses and/or distribution of the REDISTRIBUTABLE COMPONENTS.

LIMITED WARRANTY

NO WARRANTIES. Secure Plus expressly disclaims any warranty for the SOFTWARE. The SOFTWARE and any related documentation is provided "as is" without warranty of any kind, either express or implied, including, without limitation, the implied warranties or merchantability or fitness for a particular purpose. The entire risk arising out of use or performance of the SOFTWARE remains with you.

CUSTOMER REMEDIES. Each party's entire liability under this license agreement shall not exceed the price paid for the SOFTWARE. NO LIABILITY FOR CONSEQUENTIAL DAMAGES. In no event shall Secure Plus, its suppliers or you be liable for any damages whatsoever (including, without limitation, damages for

loss of business profits, business interruptions, loss of business information, or any other pecuniary loss) arising out of the use or inability to use this Secure Plus product, even if such party has been advised of the possibility of such damages. The limitations and disclaimers set forth in this section do not apply to
[a] either party's obligations of indemnity stated herein or
[b] to your material breach of your obligations under this license agreement.

DEMO. The demo versions of our products are intended for evaluation purposes only. You may not use the demo version to develop completed applications.

This agreement is protected by copyright laws and international treaty provisions. If you do not agree to the terms of the license agreement, you are not allowed to use this product or any part of it. Should you have any questions concerning this product, contact Secure Plus d.o.o.

Platforms

Windows
Mac OSX
Linux
BSD

Initialization

Before using devP2P, you must initialize devP2P library by calling libInit, and release all global resources using libFree functions.

If you want to use UPNP protocol for opening router ports (to help out devP2P establish direct connections), you should also call UPNPInit at the startup, after libInit.

Platforms

Windows
Mac OSX
Linux
BSD

libInit

Initializes devP2P library.

Syntax

C++

```
bool libInit(void);
```

The *libInit()* syntax has these parts:

Return value	Returns true on success.

Remarks

This function must be called first, once, before any usage of devP2P. It creates internal structures and classes, and prepares devP2P for usage. After calling this method, you can call Create method as many times as you wish to obtain new instances of devP2P.

When you are done with using devP2P library, call libFree function.

Platforms

Windows
Mac OSX
Linux
BSD

libFree

Frees devP2P library resources.

Syntax

C++

```
void libFree(void);
```

Remarks

libFree should be called after you have completed your work with devP2P. This function will release all internal structures. Make sure you have Destroyed all devP2P instances cleanly before freeing the library.

Platforms

Windows
Mac OSX
Linux
BSD

UPNPInit

Initializes UPNP (universal plug and play) network broadcast.

Syntax

C++

```
void upnpInit(void);
```

Remarks

UPNP is internally used by devP2P to locate possible routers on the network, and use them to open external routing ports. If such routers are found, then each instance of devP2P will be able to allocate and bind router's port for direct P2P connections with remote peers. devP2P will then use UPNP protocol as needed to open/close such ports.

Platforms

Windows
Mac OSX
Linux
BSD

Globals

Helper functions used internally by devP2P, but available to you as well.

MacToText	Converts MAC address to HEX representation.
TextToMac	Converts HEX text representation to MAC address.
VPNAdapter	Returns reference to specific virtual network adapter.
VPNAdapterCount	Total number of virtual network adapters available on the system.

Platforms

Windows
Mac OSX
Linux
BSD

MacToText function

Converts MAC address to HEX representation.

Syntax

C++

```
void MacToText(unsigned char *src, char dst[18]);
```

The *MacToText(src,dst)* syntax has these parts:

src	points to MAC address
dst	points to destination 18 byte buffer

Platforms

Windows
Mac OSX
Linux
BSD

TextToMac function

Converts HEX text representation to MAC address.

Syntax

C++

```
void TextToMac(char *src,char *dst);
```

The *TextToMac(src,dst)* syntax has these parts:

src	Souce 18 byte char array.
dst	Destination byte array where MAC address is stored.

Platforms

Windows
Mac OSX
Linux
BSD

VPNAdapter function

Returns reference to specific virtual network adapter.

Syntax

C++

```
CVPNInterface *VPNAdapter(int index);
```

The *VPNAdapter(index)* syntax has these parts:

index	Integer. Index number of the interface.
Return value	Reference to the VPNInterface object.

Remarks

VPNAdapter will return reference to virtual network driver instances that are available to use by devP2P. You can get total number of interfaces using VPNAdapterCount function.

You can share same instance of VPNAdapter between several devP2P instances, since it's needed to property route packets to more than one remote peer.

Platforms

Windows
Mac OSX
Linux
BSD

VPNAdapterCount function

Total number of virtual network adapters available on the system.

Syntax

C++

```
int VPNAdapterCount(void);
```

The *VPNAdapterCount()* syntax has these parts:

Return value	Total number of interfaces.

Remarks

To access specific interface use VPNAdapter function.

Platforms

Windows
Mac OSX
Linux
BSD

devP2P

Main devP2P object.

Create — Creates new devP2P class instance.

Destroy — Destroys current devP2P instance.

Properties

ConnectionTimeout	Determines total number of milliseconds before connection drops for inactivity.
ConnectionType	Returns type of transport with remote peer.
Events	Reference to event handlers.
LinkingRelayDelay	Total number of direct link attempts before relay is used.
LinkingRetryCount	Determines how many times retry is performed during linking stage.
LinkingRetryDelay	Determines delay time between linking attempts, in milliseconds.
MediatorAddress	Holds IP address (or hostname) of the mediator.
MediatorPort	Holds port of the mediator.
MediatorRetryCount	Determines number of retries in Search method.
MediatorRetryDelay	Number of milliseconds to wait between two retries to reach the mediator.
MyName	Holds user defined identity ID of local devP2P peer.
PeerAdapterIP	Holds IP address of remote peer for VPN.
PeerAdapterMAC	Holds MAC address of remote peer for VPN.
PeerName	Holds user defined identity ID of remote devP2P peer.
RelayLicense	Holds license information to provide to mediator for using its relay(s).
State	Returns current devP2P state.
Tag	Tag for misc usage.
TCPPort	Specifies local TCP port used for listening.
UDPPort	Specifies local UDP port used for listening.

Methods

Bandwidth	Returns calculated bandwidth usage.
BandwidthReset	Resets bandwidth calculation for specific channel.
Disconnect	Disconnects from remote devP2P peer.
ErrorText	Returns text representation of the error.

GetForward	Returns reference to CP2PForward object.
Link	Initiates connection with remote peer.
Ping	Sends internal PING packet to remote peer.
Search	Searches for remote peer using mediator.
SendData	Sends byte array message to remote peer.
SendFile	Sends file to remote peer.
SendText	Sends text message to remote peer.
SetAdapter	Assigns existing adapter for network forwarding.
SetLicenseKey	Sets your license key.
SetPassword	Determines if devP2P traffic is encrypted.
Start	Starts listening and accepting connections.
StartForward	Starts port forwarding for specific forward channel.
StateText	Returns text representation of the state.
Stop	Stops listening for connections.
StopForward	Stops specific channel forwarding.
TestBandwidth	Tests bandwidth with remote peer.
Version	Returns devP2P version information.

Events

DataReceived	Fires when data is received from remote.
FileDone	Fires when file transfer completes.
FileProgress	Fires during file transfer.
FileReceive	Fires when remote devP2P wants to send us a file.
FileSend	Fires when local devP2P starts sending file to remote.
ForwardClose	Fires when forwarding is stopped.
ForwardOpen	Fires when port forwarding is open.
LinkDone	Fires when devP2P links with remote devP2P instance.
NewUPNPMapping	Fires when new UPnP port mapping was created.
Ping	Fires when Ping packet comes from remote peer.
SearchDone	Fires when Search method completes its search for remote devP2P peer.
SearchStart	Fires when search has started.
StateChange	Fires when devP2P changes its state.
Stopped	Fires when devP2P stops working and goes offline.
TextReceived	Fires when text message arrives from remote side.

UserConnected	Fires when user connects to forwarding channel.
UserDisconnected	Fires when user disconnects from the forwarded channel.

Platforms

Windows
Mac OSX
Linux
BSD

Create method

Creates new devP2P class instance.

Syntax

C# | C++ | VB.NET

```
IntPtr devP2P.Create();
```

The *Create()* syntax has these parts:

Return value	Reference to new created CP2P instance.

Remarks

This is a static method that creates new instance of devP2P. After instance is successfully obtained and used, you should destroy it using Destroy method.

You should not delete the instance by yourself. Always use Destroy method instead.

Make sure you initialized the library first, by calling libInit function!

Code sample

C++

```
// Initialize devP2P library
devP2Plib::libInit();
devP2Plib::upnpInit();
// Give some time for UPNP to exchange packets
Sleep(200);

devP2Plib::CP2P *p1 = devP2Plib::CP2P::Create();
// ... //
```

Platforms

Windows
Mac OSX
Linux
BSD

Destroy method

Destroys current devP2P instance.

Syntax

C# | C++ | VB.NET

```
void devP2P.Destroy(IntPtr Handle);
```

Remarks

This method destroys the devP2P instance, obtained by Create method. When it's not used anymore, it is internally deleted.

Platforms

Windows
Mac OSX
Linux
BSD

Properties

ConnectionTimeout	Determines total number of milliseconds before connection drops for inactivity.
ConnectionType	Returns type of transport with remote peer.
Events	Reference to event handlers.
LinkingRelayDelay	Total number of direct link attempts before relay is used.
LinkingRetryCount	Determines how many times retry is performed during linking stage.
LinkingRetryDelay	Determines delay time between linking attempts, in milliseconds.
MediatorAddress	Holds IP address (or hostname) of the mediator.
MediatorPort	Holds port of the mediator.
MediatorRetryCount	Determines number of retries in Search method.
MediatorRetryDelay	Number of milliseconds to wait between two retries to reach the mediator.
MyName	Holds user defined identity ID of local devP2P peer.
PeerAdapterIP	Holds IP address of remote peer for VPN.
PeerAdapterMAC	Holds MAC address of remote peer for VPN.
PeerName	Holds user defined identity ID of remote devP2P peer.
RelayLicense	Holds license information to provide to mediator for using its relay(s).
State	Returns current devP2P state.
Tag	Tag for misc usage.
TCPPort	Specifies local TCP port used for listening.
UDPPort	Specifies local UDP port used for listening.

Platforms

Windows
Mac OSX
Linux
BSD

ConnectionTimeout property

Determines total number of milliseconds before connection drops for inactivity.

Type

Integer.

Syntax

C# | C++ | VB.NET

```
void devP2P.SetConnectionTimeout(IntPtr Handle, int Value);
```

The *ConnectionTimeout(Handle,value)* syntax has these parts:

Handle	Reference to the devP2P instance.
value	Integer value representing connection timeout in milliseconds.

```
int devP2P.GetConnectionTimeout(IntPtr Handle);
```

The *ConnectionTimeout(Handle)* syntax has these parts:

Handle	Reference to the devP2P instance.
Return value	Current value for ConnectionTimeout property.

Remarks

ConnectionTimeout property defines total number of milliseconds that remote peer is idle before connection is dropped for inactivity. devP2P sends "keep alive" PING packet each second after certain period of inactivity, but expects from remote side to do the same. If no packet is received for defined ConnectionTimeout value, local devP2P peer will close the connection and fire Stopped event.

Platforms

Mac OSX
Linux
BSD

ConnectionType property

Returns type of transport with remote peer.

Type

ConnectionTypes enumeration.

Syntax

C# | C++ | VB.NET

```
int devP2P.GetConnectionType(IntPtr Handle);
```

The *ConnectionType(Handle)* syntax has these parts:

Handle	Reference to the devP2P instance.
Return value	Type of established connection, from ConnectionTypes enumeration.

Remarks

This property returns type of established connection with remote peer. Typically this will be set to *ConnectionDirect*, but if RelayLicense set, and devP2P used relay to connect to remote peer, such relayed connection will be returned here so you know how you connect to remote.

Platforms

Windows
Mac OSX
Linux
BSD

Events property

Reference to event handlers.

Type

P2PEventsStruct structure.

Syntax

C++

```
P2PEventsStruct Events;
```

The *Events()* syntax has these parts:

Return value	Returns pointer to internal events structure.

Remarks

To use specific event with devP2P, you must implement your own function that has same declaration as the event, and give a reference to P2PEventsStruct for the function. P2PEventStruct members correspond to events, and default to NULL. Below in code samples is shown how to do it for some events.

This is the declaration of P2PEventsStruct

```
  typedef struct P2PEventsStruct
  {
      void (*StateChange)(CP2P *p2p, States state);
      void (*NewUPNPMapping)(CP2P *p2p, char *ExtAddress, char *IntAddress, int TCPPort, int UDPPort);
      void (*SearchStart)(CP2P *p2p, char *binds);
      void (*SearchDone)(CP2P *p2p, char *peerName, char *peerXML, char *customData, Errors error);
      void (*LinkDone)(CP2P *p2p, char *address, int port, Errors error);
      void (*Stopped)(CP2P *p2p, Errors error);
      void (*TextReceived)(CP2P *p2p, int chanid, char *text);
      void (*DataReceived)(CP2P *p2p, int chanid, char *data, int len);
      bool (*FileSend)(CP2P *p2p, int chanid, char *filename, int64 size);
      bool (*FileReceive)(CP2P *p2p, int chanid, char *filename, int64 size);
      void (*FileProgress)(CP2P *p2p, int chanid, int64 position, int64 size);
      void (*FileDone)(CP2P *p2p, int chanid, Errors error);
      bool (*ForwardOpen)(CP2P *p2p, int forwid, ForwardTypes forwardtype, char *localaddress, int localport, char
*remoteaddress, int remoteport);
      void (*ForwardClose)(CP2P *p2p, int forwid, Errors error);
      bool (*UserConnected)(CP2P *p2p, int forwid, int chanid, CP2PForwardUser *user);
      void (*UserDisconnected)(CP2P *p2p, int forwid, int chanid, CP2PForwardUser *user, Errors error);
      void (*Ping)(CP2P *p2p);
  } P2PEventsStruct;
```

Code sample

C++

```
void p2p_StateChange(devP2Plib::CP2P *p2p, devP2Plib::States state)
{
   printf("%s] State changed to %s\r\n", p2p->MyName, p2p->StateText(state));
}
```

```
void p2p_SearchStart(devP2Plib::CP2P *p2p, char *binds)
{
   printf("%s] started searching for %s at %s (%d)\r\n", p2p->MyName, p2p->PeerName, p2p-
>MediatorAddress, p2p->MediatorPort);
}
void p2p_SearchDone(devP2Plib::CP2P *p2p, char *peerName, char *peerXML, char *customData,
devP2Plib::Errors error)
{
   printf("%s] Search finished with error %d %s\r\n", p2p->MyName, error, p2p->ErrorText(error));
   if (!error)
       p2p->Link(peerXML);
}
/* ...... */

int main(int argc, char **argv)
{
   devP2Plib::CP2P *p1 = devP2Plib::CP2P::Create();
/* ..... */
   p1->Events.SearchDone = p2p_SearchDone;
   p1->Events.SearchStart = p2p_SearchStart;
   p1->Events.StateChange = p2p_StateChange;
/* ..... */
}
```

Platforms

Windows
Mac OSX
Linux
BSD

LinkingRelayDelay property

Total number of direct link attempts before relay is used.

Type

Integer.

Syntax

C# | C++ | VB.NET

```
void devP2P.SetLinkingRelayDelay(IntPtr Handle, int Value);
```

The *LinkingRelayDelay(Handle,value)* syntax has these parts:

Handle	Reference to the devP2P instance.
value	Number of linking attempts.

```
int devP2P.GetLinkingRelayDelay(IntPtr Handle);
```

The *LinkingRelayDelay(Handle)* syntax has these parts:

Handle	Reference to the devP2P instance.
Return value	Current value of LinkingRelayDelay property.

Remarks

Assuming you set your RelayLicense property, and are allowed to use the relay from the mediator, this property defines how many linking packets will be sent to remote peer trying to establish direct connection. If connection is not established after *LinkingRetryCount* packets, devP2P will send packets to relay as well, and most probably succeed in establishing relayed connection.

Note that relayed connection can be twice slower as direct one, since extra hop is used for transferring the data.

Platforms

Windows
Mac OSX
Linux
BSD

LinkingRetryCount property

Determines how many times retry is performed during linking stage.

Type

Integer.

Syntax

C# | C++ | VB.NET

```
void devP2P.SetLinkingRetryCount(IntPtr Handle, int Value);
```

The *LinkingRetryCount(Handle,value)* syntax has these parts:

Handle	Reference to the devP2P instance.
value	Total number of retries to exchange initial linking packet with remote peer.

```
int devP2P.GetLinkingRetryCount(IntPtr Handle);
```

The *LinkingRetryCount(Handle)* syntax has these parts:

Handle	Reference to the devP2P instance.
Return value	Current value of LinkingRetryCount property.

Remarks

When Link is called, devP2P sends packet to remote peer trying to establish the connection. This property defines how many packets will be sent before LinkDone event fires with an error.

Platforms

Windows
Mac OSX
Linux
BSD

LinkingRetryDelay property

Determines delay time between linking attempts, in milliseconds.

Syntax

C# | C++ | VB.NET

```
void devP2P.SetLinkingRetryDelay(IntPtr Handle, int Value);
```

The *LinkingRetryDelay(Handle,value)* syntax has these parts:

Handle	Reference to the devP2P instance.
value	Millisecond delay between two linking attempts.

```
int devP2P.GetLinkingRetryDelay(IntPtr Handle);
```

The *LinkingRetryDelay(Handle)* syntax has these parts:

Handle	Reference to the devP2P instance.
Return value	Current vlalue of LinkingRetryDelay property.

Remarks

When Link is called, devP2P sends packet to remote peer trying to establish the connection. This property defines how much will devP2P wait between each two retries in sending packets, in milliseconds.

Platforms

Windows
Mac OSX
Linux
BSD

MediatorAddress property

Holds IP address (or hostname) of the mediator.

Type

String.

Syntax

C# | C++ | VB.NET

```
void devP2P.GetMediatorAddress(IntPtr Handle, StringBuilder buffer);
```

The *MediatorAddress(Handle,buffer)* syntax has these parts:

Handle	Reference to the devP2P instance.
buffer	Buffer where output is stored.

```
void devP2P.SetMediatorAddress(IntPtr Handle, string Value);
```

The *MediatorAddress(Handle,Value)* syntax has these parts:

Handle	Reference to the devP2P instance.
Value	New mediator address to set.

Remarks

Set this property to hostname of mediator that is used with Search method, together with MediatorPort property. devP2P will send mediator requests to that IP/Port to locate and possibly request connection with remote peer.

UDP connection is used for mediator, to help with UDP hole punching for direct peer-to-peer connection. Even if you use only TCP protocol in Start method, mediator can be used to obtain information about announced TCP ports open by remote peer.

Note that devP2P will remember this property through address reference, so keep your buffers static and valid as long as devP2P needs it. devP2P will not free or touch allocated memory in any way.

Platforms

Windows
Mac OSX
Linux
BSD

MediatorPort property

Holds port of the mediator.

Type

Integer.

Syntax

C# | C++ | VB.NET

```
void devP2P.SetMediatorPort(IntPtr Handle, int Value);
```

The *MediatorPort(Handle,value)* syntax has these parts:

Handle	Reference to the devP2P instance.
value	Port where mediator listens.

```
int devP2P.GetMediatorPort(IntPtr Handle);
```

The *MediatorPort(Handle)* syntax has these parts:

Handle	Reference to the devP2P instance.
Return value	Integer value representing mediator's port.

Remarks

Set this property to port where mediator listens for Search method, together with MediatorAddress property. devP2P will send mediator requests to that IP/Port to locate and possibly request connection with remote peer.

UDP connection is used for mediator, to help with UDP hole punching for direct peer-to-peer connection. Even if you use only TCP protocol in Start method, mediator can be used to obtain information about announced TCP ports open by remote peer.

Platforms

Windows
Mac OSX
Linux
BSD

MediatorRetryCount property

Determines number of retries in Search method.

Type

Integer.

Syntax

C# | C++ | VB.NET

```
void devP2P.SetMediatorRetryCount(IntPtr Handle, int Value);
```

The *MediatorRetryCount(Handle,value)* syntax has these parts:

Handle	Reference to the devP2P instance.
value	Total number of retries to reach the mediator.

```
int devP2P.GetMediatorRetryCount(IntPtr Handle);
```

The *MediatorRetryCount(Handle)* syntax has these parts:

Handle	Reference to the devP2P instance.
Return value	Current value of MediatorRetryCount property.

Remarks

After Search is called, devP2P sends packets to mediator searching for remote peer. This property defines how many packets will be sent to mediator before SearchDone is returned with an error.

MediatorRetryDelay property

Number of milliseconds to wait between two retries to reach the mediator.

Type

Integer.

Syntax

C# | C++ | VB.NET

```
void devP2P.SetMediatorRetryDelay(IntPtr Handle, int Value);
```

The *MediatorRetryDelay(Handle,value)* syntax has these parts:

Handle	Reference to the devP2P instance.
value	Millisecond delay between two retry attempts.

```
int devP2P.GetMediatorRetryCount(IntPtr Handle);
```

The *MediatorRetryDelay(Handle)* syntax has these parts:

Handle	Reference to the devP2P instance.
Return value	Current value of MediatorRetryDelay property.

Remarks

After Search is called, devP2P sends packets to mediator searching for remote peer. This property defines delay between two packets, in milliseconds. MediatorRetryCount property defines how many packets will be sent before SearchDone is returned with an error.

Platforms

Windows
Mac OSX
Linux
BSD

MyName property

Holds user defined identity ID of local devP2P peer.

Type

String.

Syntax

C# | C++ | VB.NET

```
void devP2P.SetMyName(IntPtr Handle, string Value);
```

The *MyName(Handle,value)* syntax has these parts:

Handle	Reference to the devP2P instance.
value	Buffer with new name for local peer.

```
void devP2P.GetMyName(IntPtr Handle, StringBuilder buffer);
```

The *MyName(Handle)* syntax has these parts:

Handle	Reference to the devP2P instance.
Return value	Current name for local peer.

Remarks

MyName property holds ID that is shown to remote side upon successful connection. Value isn't really important, it's just basic exchanged information about peers when they connect, so you know in LinkDone event that you actually got connected to peer you wanted to. However, to be 100% sure in remote peer's identity, you should use same secret password on both sides since complete traffic is encrypted with it.

However, if Search method is used, MyName can be very important since it announces your ID to the mediator - and remote peer will search for you using that ID value. Note that devP2P will remember this property through address reference, so keep your buffers static and valid as long as devP2P needs it. devP2P will not free or touch allocated memory in any way.

Platforms

Windows
Mac OSX
Linux
BSD

PeerAdapterIP property

Holds IP address of remote peer for VPN.

Type

Unsigned integer.

Syntax

C++

```
unsigned int PeerAdapterIP;
```

Remarks

PeerAdapterIP contains IP address of remote peer when full network traffic is routed through devP2P. It is necessary for proper redirection because Windows/Linux sockets needs remote's IP address (together with his MAC address) so raw packets are sent to correct destination.

Usually you will leave this property empty, since it will be filled when LinkDone event fires. However, devP2P uses this information to answer Windows sockets ARP requests, so you can even speed up packet exchange startup if you fill this value by yourself - just make sure correct IP is entered here.

In order to obtain local IP for each adapter, you should check VPNInterface's LocalIP property.

Code sample

C++

```
// To obtain string representation of the address, use code like this:
char buff[1024];
sprintf(buff, "%s", inet_ntoa(peer1->PeerAdapterIP));
```

Platforms

Windows
Mac OSX
Linux
BSD

PeerAdapterMAC property

Holds MAC address of remote peer for VPN.

Type

String.

Syntax

C++

```
unsigned char PeerAdapterMAC[6];
```

Remarks

PeerAdapterMAC holds MAC address of remote peer's VPN interface. It is needed by Windows/Linux sockets to determine where to send raw network packets. Each Ethernet card can accept several IP addresses (and perhaps route them to other devices), so they need to 'announce' list of IP addresses that they will collect packets for. devP2P will fill this property when LinkDone event fires. However, if you wish to speed up the process, you can fill this value by yourself on startup, so devP2P can answer Windows/Linux sockets' questions with correct packets even sooner than VPN is established with remote peer. Typical MAC address looks like this: 00:ff:e5:41:ae:a7.

Together with this property, you can also set PeerAdapterIP property. If this property remains empty, devP2P will fill it for you.

In order to obtain local MAC for each adapter, you should check VPNInterface's LocalMAC property.

You can use MacToText to get string representation of the MAC address.

Platforms

Windows
Mac OSX
Linux
BSD

PeerName property

Holds user defined identity ID of remote devP2P peer.

Type

String.

Syntax

C# | C++ | VB.NET

```
void devP2P.SetPeerName(IntPtr Handle, string Value);
```

The *PeerName(Handle,value)* syntax has these parts:

Handle	Reference to the devP2P instance.
value	Buffer with new name for remote peer.

```
void devP2P.GetPeerName(IntPtr Handle, StringBuilder buffer);
```

The *PeerName(Handle)* syntax has these parts:

Handle	Reference to the devP2P instance.
Return value	Current name for remote peer.

Remarks

PeerName property holds ID that of the peer you're connecting to. Value isn't really important for devP2P protocol, but it is used to Search remote peer on the mediator, and when connection arrives to test if it matches what is offered by remote side. However, to be 100% sure in remote peer's identity, you should use same secret password on both sides since complete traffic is encrypted with it.

Note that devP2P will remember this property through address reference, so keep your buffers static and valid as long as devP2P needs it. devP2P will not free or touch allocated memory in any way.

Platforms

Windows
Mac OSX
Linux
BSD

RelayLicense property

Holds license information to provide to mediator for using its relay(s).

Type

String.

Syntax

C# | C++ | VB.NET

```
void devP2P.SetRelayLicense(IntPtr Handle, string Value);
```

The *RelayLicense(Handle,Value)* syntax has these parts:

Handle	Reference to the devP2P instance.
Value	Licensing data.

```
void devP2P.GetRelayLicense(IntPtr Handle,  StringBuilder buffer);
```

The *RelayLicense(Handle)* syntax has these parts:

Handle	Reference to the devP2P instance.
Return value	Licensing data.

Remarks

RelayLicense property is used when Search method is called, and you want to use relaying features from the mediator. This data is provided by your mediator in advance, and (depending on mediator's setup) it is required to use advanced mediator relaying features.

Licensing for relays is included in devP2P to prevent from 3rd parties to use your mediator for "expensive" relaying features, in terms of CPU and bandwidth usage. You will use this data on the mediator's side to allow specific devP2P instances to connect to your own relays or deny them.

Note that devP2P will remember this property through address reference, so keep your buffers static and valid as long as devP2P needs it. devP2P will not free or touch allocated memory in any way.

Platforms

Windows
Mac OSX
Linux
BSD

State property

Returns current devP2P state.

Type

States enumeration.

Syntax

C# | C++ | VB.NET

```
int devP2P.GetState(IntPtr Handle);
```

The *State(Handle)* syntax has these parts:

Handle	Reference to the devP2P instance.
Return value	Current state from States enumeration.

Remarks

State property returns current devP2P state. If devP2P is completely idle, State will hold *StateStopped* value. As soon as devP2P starts with some activity, StateChange event will fire where you can keep track on devP2P's behavior.

Platforms

Windows
Mac OSX
Linux
BSD

Tag property

Tag for misc usage.

Type

Object.

Syntax

C# | C++ | VB.NET

```
void devP2P.SetTag(IntPtr Handle, object Value);
```

The *Tag(Handle,value)* syntax has these parts:

Handle	Reference to the devP2P instance.
value	Object that holds your data.

```
object devP2P.GetTag(IntPtr Handle);
```

The *Tag(Handle)* syntax has these parts:

Handle	Reference to the devP2P instance.
Return value	Object that holds your data.

Remarks

You can use this property to store pointer to your own custom data that will be kept by devP2P. devP2P will not interfere with this value in any way (it will not, for example, try to free that memory).

Platforms

Windows
Mac OSX
Linux
BSD

TCPPort property

Specifies local TCP port used for listening.

Type

Integer.

Syntax

C# | C++ | VB.NET

```
void devP2P.SetTCPPort(IntPtr Handle, int Value);
```

The *TCPPort(Handle,value)* syntax has these parts:

Handle	Reference to the devP2P instance.
value	Integer value representing local TCP port.

```
int devP2P.GetTCPPort(IntPtr Handle);
```

The *TCPPort(Handle)* syntax has these parts:

Handle	Reference to the devP2P instance.
Return value	Integer value representing local TCP port.

Remarks

TCPPort property defines on local TCP port where devP2P listens and accepts connections from remote peer. You can predefine it before calling Start method, in which case devP2P will force using your port. You can also set it to 0, in which case devP2P will allocate first free port (as provided by the system) and fill this property with allocated port.

If you're not interested which port will be allocated in Start method, it is wise to set this property to zero each time before calling the Start method, since on subsequent Start method calls devP2P could reuse previously filled port value.

Platforms

Windows
Mac OSX
Linux
BSD

UDPPort property

Specifies local UDP port used for listening.

Type

Integer.

Syntax

C# | C++ | VB.NET

```
void devP2P.SetUDPPort(IntPtr Handle, int Value);
```

The *UDPPort(Handle,value)* syntax has these parts:

Handle	Reference to the devP2P instance.
value	Integer value representing local UDP port.

```
int devP2P.GetUDPPort(IntPtr Handle);
```

The *UDPPort(Handle)* syntax has these parts:

Handle	Reference to the devP2P instance.
Return value	Integer value representing local UDP port.

Remarks

UDPPort property defines on local UDP port where devP2P listens and accepts connections from remote peer. You can predefine it before calling Start method, in which case devP2P will force using your port. You can also set it to 0, in which case devP2P will allocate first free port (as provided by the system) and fill this property with allocated port.

If you're not interested which port will be allocated in Start method, it is wise to set this property to zero each time before calling the Start method, since on subsequent Start method calls devP2P could reuse previously filled port value.

Platforms

Windows
Mac OSX
Linux
BSD

Methods

Bandwidth	Returns calculated bandwidth usage.
BandwidthReset	Resets bandwidth calculation for specific channel.
Disconnect	Disconnects from remote devP2P peer.
ErrorText	Returns text representation of the error.
GetForward	Returns reference to CP2PForward object.
Link	Initiates connection with remote peer.
Ping	Sends internal PING packet to remote peer.
Search	Searches for remote peer using mediator.
SendData	Sends byte array message to remote peer.
SendFile	Sends file to remote peer.
SendText	Sends text message to remote peer.
SetAdapter	Assigns existing adapter for network forwarding.
SetLicenseKey	Sets your license key.
SetPassword	Determines if devP2P traffic is encrypted.
Start	Starts listening and accepting connections.
StartForward	Starts port forwarding for specific forward channel.
StateText	Returns text representation of the state.
Stop	Stops listening for connections.
StopForward	Stops specific channel forwarding.
TestBandwidth	Tests bandwidth with remote peer.
Version	Returns devP2P version information.

Platforms

Windows
Mac OSX
Linux
BSD

Bandwidth method

Returns calculated bandwidth usage.

Type

Float.

Syntax

C# | C++ | VB.NET

```
float devP2P.Bandwidth(IntPtr Handle, int chanid);
```

The *Bandwidth(Handle,chanid)* syntax has these parts:

Handle	Reference to the devP2P instance.
chanid	Integer that specifies channel index.
Return value	Float value representing megabytes per second bandwidth usage.

Remarks

This method returns calculated bandwidth usage for specific channel it. You can use it during, for example, file tranfers to monitor file transfer speed, since devP2P will autocalculate it for you. You can call BandwidthReset method to reset calculation just before file transfer start, to get accurate values.

Code sample

C++

```
void p2p_FileProgress(devP2Plib::CP2P *p2p, int chanid, int64 position, int64 size)
{
   printf("%s] Progress %ld/%ld (%d%%) bandwidth=%5.2f MB/sec\r\n", p2p->MyName, (long)position,
(long)size, (int)((position*100)/size), p2p->Bandwidth(chanid));
}
```

Platforms

Windows
Mac OSX
Linux
BSD

BandwidthReset method

Resets bandwidth calculation for specific channel.

Syntax

C# | C++ | VB.NET

```
void devP2P.BandwidthReset(IntPtr Handle, int chanid);
```

The *BandwidthReset(Handle,chanid)* syntax has these parts:

Handle	Reference to the devP2P instance.
chanid	Integer that specifies channel index.

Remarks

Calling this function causes internal bandwidth calculation to resets to 0. You should call it before you start file transfer, if you want to get accurate bandwidth results.

Platforms

Windows
Mac OSX
Linux
BSD

Disconnect method

Disconnects from remote devP2P peer.

Syntax

C# | C++ | VB.NET

```
void devP2P.Disconnect(IntPtr Handle);
```

The *Disconnect(Handle)* syntax has these parts:

Handle	Reference to the devP2P instance.

Remarks

Disconnect method will stop sending/receiving TCP/UDP traffic between local and remote devP2P, thus "closing the socket" and breaking the connection and all ongoing transfers. All channels will be closed, and no new channels will be accepted until connection is established again.

When Disconnect is called, component will not be idle. It will still be in Listening state accepting remote connections (since any devP2P side can actually initiate it). If you want to shut down devP2P completely, you should use the Stop method.

If possible, remote side will receive disconnect packet so it can close its connection as well. This actually depends if network stack was fast enough to send out disconnect packet before socket is closed (applies to UDP transport).

Platforms

Windows
Mac OSX
Linux
BSD

ErrorText method

Returns text representation of the error.

Type

String.

Syntax

C# | C++ | VB.NET

```
void devP2P.ErrorText(IntPtr Handle, int errorno, StringBuilder buffer);
```

The *ErrorText(Handle,error,buffer)* syntax has these parts:

Handle	Reference to the devP2P instance.
error	Errors value.
buffer	String buffer where error text will be stored.

Remarks

This method will return text representation of the error, in English. You can use it to show user-friendly information in your application.

Platforms

Windows
Mac OSX
Linux
BSD

GetForward method

Returns reference to CP2PForward object.

Type

CP2PForward object.

Syntax

C# | C++ | VB.NET

```
IntPtr devP2P.GetForward(IntPtr Handle, int index);
```

The *GetForward(Handle,index)* syntax has these parts:

Handle	Reference to the devP2P instance.
index	Index of the forwarded channel you want to access.
Return value	CP2PForward object reference.

Remarks

GetForward will return pointer to CP2PForward class, so you can access specific elements of the forwarded channel - such as LocalAddress, Type, etc..

Platforms

Windows
Mac OSX
Linux
BSD

Link method

Initiates connection with remote peer.

Type

Integer.

Syntax

C# | C++ | VB.NET

```
int devP2P.Link(IntPtr Handle, string peerXML);
```

The *Link(Handle,peerXML)* syntax has these parts:

Handle	Reference to the devP2P instance.
peerXML	XML containing list of IP address where remote listens.
Return value	0 (ErrorNone) if success, otherwise negative value from Errors enumeration.

Remarks

The Link method will try to connect to remote devP2P. Connection is attempted both with UDP and TCP packets (based on Start method arguments). Besides persistent TCP connection, devP2P implements "reliable UDP" connection between two peers, and forwards all traffic through it, whatever succeeds first.

In order for Link to be successful, you should provide peerXML details, structured XML containing IP and Port details, where component should try to connect to. Typically, you will get this value in SearchDone event, but you can fill it by yourself if you know where remote peer listens, and want to try to establish direct connection with known parameters.

If local devP2P connects to remote devP2P, LinkDone event will be fired. Initial "handshake" will be performed prior to firing this event. All packets that are sent to remote side are encrypted using AES algorithm using key set in the SetPassword method. Both sides, of course, must use same password otherwise remote peer's data is unreadable.

Once LinkDone event is fired, you can, for example, send text, files, or get reference to any of free forward channels (through GetForward method), set up it's properties, and call StartForward to start forwarding traffic.

Typically, peerXML will look like this - specifies type of connection (TCP/UDP), IP and port, and possibly type of connection when specified:

```
<UDP Type="Mediator" IP="81.127.37.44" Port="52133"/>
<UDP IP="192.168.42.1" Port="61517"/>
<TCP IP="192.168.42.1" Port="19815"/>
<UDP IP="192.168.111.1" Port="61517"/>
<TCP IP="192.168.111.1" Port="19815"/>
<UDP IP="25.145.193.136" Port="61517"/>
<TCP IP="25.145.193.136" Port="19815"/>
<UDP IP="192.168.192.8" Port="61517"/>
<TCP IP="192.168.192.8" Port="19815"/>
<UDP IP="81.127.37.44" Port="61517"/>
<TCP IP="81.127.37.44" Port="19815"/>
<TCP Type="Relay" IP="1.2.3.4" Port="123"/>
```

XML elements are pretty much self explanatory. Besides setting UDP/TCP xml entry, you must define IP and Port attributes. If Type="Mediator" is set, devP2P will try to perform "UDP Hole Punching" on specified IP/Port to penetrate through remote firewall. If Type="Relay" then devP2P will not attempt to establish connection with it immediately, but will wait first few direct attempts, and when number of attempts defined by LinkingRelayDelay passes, it will attempt to connect through relay too.

Platforms

Windows
Mac OSX
Linux
BSD

Ping method

Sends internal PING packet to remote peer.

Type

Integer.

Syntax

C# | C++ | VB.NET

```
int devP2P.Ping(IntPtr Handle);
```

The *Ping(Handle)* syntax has these parts:

Handle	Reference to the devP2P instance.
Return value	0 (ErrorNone) if success, otherwise negative value from Errors enumeration.

Remarks

This method initiates sending of internal Ping packet to remote. As a result, on remote Ping event will fire. It is usually used to check if remote side is still connected.

Platforms

Windows
Mac OSX
Linux
BSD

Search method

Searches for remote peer using mediator.

Type

Integer.

Syntax

C# | C++ | VB.NET

```
int devP2P.Search(IntPtr Handle, string customdata);
```

The *Search(Handle,customdata)* syntax has these parts:

Handle	Reference to the devP2P instance.
customdata	String buffer to send to remote peer.
Return value	0 (ErrorNone) if success, otherwise negative value from Errors enumeration.

Remarks

Before calling Search method, make sure you set peer related properties MyName and PeerName, so devP2P knows who to search for. Also, make sure you have set MediatorAddress and MediatorPort so we have a place to send search request to.

Search method is used to locate remote peers whose IP/Port is not known, and needs be be located. Usually, the "man in the middle", who is publicly available, is needed so both peers can announce their presence to the mediator. When mediator finds a match, he replies to both peers with other peer's details, and finishes communication. Mediator is **NOT used to relay** the data, unless specifically setup to do so.

When search completes, SearchDone event will be fired with search results, being successful or not. If successful, then you will receive information about remote peer's possible IP/Port combinations where connection can be attempted.

You can use customdata to send to remote peer, which can be used by remote before he attempts to Link with you.

Platforms

Windows
Mac OSX
Linux
BSD

SendData method

Sends byte array message to remote peer.

Type

Integer.

Syntax

C# | C++ | VB.NET

```
int devP2P.SendData(IntPtr Handle, int chanid, string data, int len, bool reliable);
```

The *SendData(Handle,chanid,data,len,reliable)* syntax has these parts:

Handle	Reference to the devP2P instance.
chanid	Integer that specifies channel index.
data	String with data.
len	Total number of bytes to send.
reliable	Determines if packet should be delivered in reliable way. Defaults to true.
Return value	0 (ErrorNone) if success, otherwise negative value from Errors enumeration.

Remarks

SendData will send byte array to remote peer. On peer's side, DataReceived event will fire when byte array arrives. You can use this method only after connection with remote peer is established, of course.

You should select one of 1024 available channels to send this message. Make sure that channel is not already used by file transfer, or channel forwarding. You can send more than one message through the channel, so usually you can pick channel 0 for messages of any kind.

If you did not specify reliable method, it is possible packet never reaches the destination. This makes sense for information that is not important and possibly will be resent (such as some status information you want to provide to remote side).

You will not get any confirmation that your message is delivered.

Platforms

Windows
Mac OSX
Linux
BSD

SendFile method

Sends file to remote peer.

Type

Integer.

Syntax

C# | C++ | VB.NET

```
int devP2P.SendFile(IntPtr Handle, int chanid, string localfilename, string remotefilename, bool doresume);
```

The *SendFile(Handle,chanid,localfilename,remotefilename,doresume)* syntax has these parts:

Handle	Reference to the devP2P instance.
chanid	Integer that specifies channel index.
localfilename	Full path to local file.
remotefilename	Path where to save file on remote.
doresume	Determines if file transfer should be resumed from previous position.
Return value	0 (ErrorNone) if success, otherwise negative value from Errors enumeration.

Remarks

This method will initiate sending file to remote peer. Remote side will receive FileSend event (to decide if it wants to accept the file), then one or more FileProgress events, and finally FileDone when transfer finishes.

You should use one of available 1024 channels to send this file, just make sure this channel isn't already busy with transferring other file, or channel forwarding.

Platforms

Windows
Mac OSX
Linux
BSD

SendText method

Sends text message to remote peer.

Type

Integer.

Syntax

C# | C++ | VB.NET

```
int devP2P.SendText(IntPtr Handle, int chanid, string text, bool reliable);
```

The *SendText(Handle,chanid,text,reliable)* syntax has these parts:

Handle	Reference to the devP2P instance.
chanid	Integer that specifies channel index.
text	String buffer to send.
reliable	Determines if packet should be delivered in reliable way. Defaults to true.
Return value	0 (ErrorNone) if success, otherwise negative value from Errors enumeration.

Remarks

SendText will send NULL terminated text buffer to remote peer. On peer's side, TextReceived event will fire when text arrives. You can use this method only after connection with remote peer is established, of course.

You should select one of 1024 available channels to send this message. Make sure that channel is not already used by file transfer, or channel forwarding. You can send more than one message through the channel, so usually you can pick channel 0 for messages of any kind.

If you did not specify reliable method, it is possible packet never reaches the destination. This makes sense for information that is not important and possibly will be resent (such as some status information you want to provide to remote side).

You will not get any confirmation that your message is delivered.

Platforms

Windows
Mac OSX
Linux
BSD

SetAdapter method

Assigns existing adapter for network forwarding.

Type

Integer.

Syntax

C++

```
int SetAdapter(CVPNInterface *adapter);
```

The *SetAdapter(adapter)* syntax has these parts:

adapter	Reference to CVPNAdapter, received from VPNAdapter function.
Return value	0 (ErrorNone) if success, otherwise negative value from Errors enumeration.

Remarks

SetAdapter specifies one or local adapters to be monitored by devP2P, in order to capture network packets and forwards them to remote peer. It is used in combination with PeerAdapterIP and PeerAdapterMAC which are provided by remote, if it also supports network packet forwarding features.

When this method is used, devP2P will capture all network packets for destination IP address and route it to remote peer, who will then give it to the system - just as it has been received from the network card. This way you get **real VPN** between your two devP2P peers. In order for VPN to work, make sure selected IP addresses of local adapter and remote peer's adapter are in same network range, and MAC addresses are setup correctly (different, unique on the network, never changed).

You can test if network packet forwarding works after you are Linkeda> with your peer with simple command prompt 'ping' command. It does not matter what OS is running on remote.

Platforms

Windows
Mac OSX
Linux
BSD

SetLicenseKey method

Sets your license key.

Syntax

C# | C++ | VB.NET

```
void devP2P.SetLicenseKey(IntPtr Handle, string Value);
```

The *SetLicenseKey(Handle,value)* syntax has these parts:

Handle	Reference to the devP2P instance.
value	Your license key.

Remarks

SetLicenseKey "unlocks" instance of devP2P to work beyond evaluation period. You can distribute devP2P with your application when your license key is set, so it does not show any kind of 'nag screens' or notifications related to evaluation period.

Platforms

Windows
Mac OSX
Linux
BSD

SetPassword method

Determines if devP2P traffic is encrypted.

Syntax

C# | C++ | VB.NET

```
void devP2P.SetPassword(IntPtr Handle, int encryption, string password);
```

The *SetPassword(Handle,encryption,password)* syntax has these parts:

Handle	Reference to the devP2P instance.
encryption	Encryptions value, defines encryption algorithm.
password	String with the password. Size depends on selected encryption algorithm.

Remarks

SetPassword method specifies devP2P to use some encryption algorithm and given password to encrypt all network traffic between two peers. Both sides must choose same encryption, and same passwords, in order for communication to be established.

Besides having secure connection, this also ensures 3rd party does not connect to your devP2P peer, since with wrong password it will not be able to establish connection in the first place.

Platforms

Windows
Mac OSX
Linux
BSD

Start method

Starts listening and accepting connections.

Type

Integer.

Syntax

C# | C++ | VB.NET

```
int devP2P.Start(IntPtr Handle, int protocol);
```

The *Start(Handle,protocol)* syntax has these parts:

Handle	Reference to the devP2P instance.
protocol	Protocols value, defines which transport protocol is allowed.
Return value	0 (ErrorNone) if success, otherwise negative value from Errors enumeration.

Remarks

Start method enabled devP2P to accept incoming connections. It is first method you will run after you set MyName and PeerName properties. At this point connection can already be established even you did not call Search or Link methods - because other peer may have done so and knows IP/Port of your peer.

You can select here if you will allow usage of TCP and/or UDP ports for transport communication. Usually you will allow both. TCP is fast and reliable, and often is preferred. However, UDP penetrates firewalls in many cases, and will work where TCP doesn't. Our implementation of 'reliable UDP' matches speed of TCP so you will have usually no losses if UDP connection is established (except in rare cases where network connection is really bad). You can also select TCPPort and UDPPort where devP2P listens, or leave 0 so they are autoselected.

After Start is called, you can call Search to locate remote peer, or Link to it when you know his IP/Port where you should attempt connection. After you're done with using devP2P, you should call Stop method.

Platforms

Windows
Mac OSX
Linux
BSD

StartForward method

Starts port forwarding for specific forward channel.

Type

Integer.

Syntax

C# | C++ | VB.NET

```
int devP2P.StartForward(IntPtr Handle, int index);
```

The *StartForward(Handle,index)* syntax has these parts:

Handle	Reference to the devP2P instance.
index	Index of the forwarded channel.
Return value	0 (ErrorNone) if success, otherwise negative value from Errors enumeration.

Remarks

devP2P supports 32 forwarding channels you can setup, and each of them can provide (in theory) unlimited number of connection. For example, one such channel would be to forward local port 80 to remote peer's port 80. Once such channel is setup and started, any number of users can connect to local port, which will be redirected to remote. All of these connections belong to one channel.

By default all channels are setup to do nothing. When you want to enable one of forwarding channels, you should get reference to CP2PForward object at given index (allowed 0-31) using GetForward method, set up it's properties, and then call this method to enable it and start it. When you're done using this channel, you can call StopForward.

As users connect and start using the channel, devP2P will always assign first free channel higher than 100 to route packets for the user. When connection with that user is closed, channel will be freed. It is, because of this, suggested NOT to use channels higher than 100 for your own data and text messages.

Code sample

C++

```
devP2Plib::CP2PForward *f = devP2P->GetForward(0);
if (f)
{
   f->Type = devP2Plib::TCPSocksProxy;
   strcpy(f->LocalAddress, "127.0.0.1");
   f->LocalPort = 1080;
   success = p1->StartForward(0);
}
```

Platforms

Windows
Mac OSX
Linux
BSD

StateText method

Returns text representation of the state.

Type

String.

Syntax

C# | C++ | VB.NET

```
void devP2P.StateText(IntPtr Handle, int state, StringBuilder buffer);
```

The *StateText(Handle,state,buffer)* syntax has these parts:

Handle	Reference to the devP2P instance.
state	States value.
buffer	String buffer where state text will be stored.

Remarks

This method will return text representation of the state, in English. You can use it to show user-friendly information in your application. Usually you will use this method in StateChange event.

Platforms

Windows
Mac OSX
Linux
BSD

Stop method

Stops listening for connections.

Syntax

C# | C++ | VB.NET

```
void devP2P.Stop(IntPtr Handle);
```

The *Stop(Handle)* syntax has these parts:

Handle	Reference to the devP2P instance.

Remarks

Stop method will stop all activity of devP2P. It will disconnect remote peer (if connected), and will stop listening and accepting further connections. At this point you can change devP2P's properties and it's behavior (such as change password or MyName) and Start it again.

Platforms

Windows
Mac OSX
Linux
BSD

StopForward method

Stops specific channel forwarding.

Type

Integer.

Syntax

C# | C++ | VB.NET

```
int devP2P.StopForward(IntPtr Handle, int index);
```

The *StopForward(Handle,index)* syntax has these parts:

Handle	Reference to the devP2P instance.
index	Index of the forwarded channel.
Return value	0 (ErrorNone) if success, otherwise negative value from Errors enumeration.

Remarks

StopForward will close all forwarded connections that belong to the channel, and will not accept any new connections. It will also stop listening on specified ports that we assigned to the channel in StartForward method.

Platforms

Windows
Mac OSX
Linux
BSD

TestBandwidth method

Tests bandwidth with remote peer.

Type

Integer.

Syntax

C# | C++ | VB.NET

```
int devP2P.TestBandwidth(IntPtr Handle, int chanid, long size);
```

The *TestBandwidth(Handle,chanid,size)* syntax has these parts:

Handle	Reference to the devP2P instance.
chanid	Integer that specifies channel index.
size	Total size of data to transmit to remote.
Return value	0 (ErrorNone) if success, otherwise negative value from Errors enumeration.

Remarks

TestBandwith is used to measure speed with remote peer. It is almost the same as SendFile, but random data is sent to remote, and remote does not store it anywhere. It can be used to quickly check how good is your connection with remote peer. You can use Bandwidth method to obtain calculated bandwidth values.

Platforms

Windows
Mac OSX
Linux
BSD

Version method

Returns devP2P version information.

Type

String.

Syntax

C# | C++ | VB.NET

```
void devP2P.GetVersion(IntPtr Handle, StringBuilder buffer);
```

The *Version(Handle,buffer)* syntax has these parts:

Handle	Reference to the devP2P instance.
buffer	String buffer where version data will be stored.

Remarks

This method will return version details of your devP2P instance.

Platforms

Windows
Mac OSX
Linux
BSD

Events

DataReceived	Fires when data is received from remote.
FileDone	Fires when file transfer completes.
FileProgress	Fires during file transfer.
FileReceive	Fires when remote devP2P wants to send us a file.
FileSend	Fires when local devP2P starts sending file to remote.
ForwardClose	Fires when forwarding is stopped.
ForwardOpen	Fires when port forwarding is open.
LinkDone	Fires when devP2P links with remote devP2P instance.
NewUPNPMapping	Fires when new UPnP port mapping was created.
Ping	Fires when Ping packet comes from remote peer.
SearchDone	Fires when Search method completes its search for remote devP2P peer.
SearchStart	Fires when search has started.
StateChange	Fires when devP2P changes its state.
Stopped	Fires when devP2P stops working and goes offline.
TextReceived	Fires when text message arrives from remote side.
UserConnected	Fires when user connects to forwarding channel.
UserDisconnected	Fires when user disconnects from the forwarded channel.

Platforms

Windows
Mac OSX
Linux
BSD

DataReceived event

Fires when data is received from remote.

Syntax

C# | C++ | VB.NET

```
delegate void devP2P.DataReceivedEvent(IntPtr Handle, int chanid, String data, int len);
```

The *DataReceived(Handle,chanid,data,len)* syntax has these parts:

Handle	Reference to the devP2P instance.
chanid	Index of the channel where data arrived.
data	String with incoming data.
len	Size of incoming data.

Remarks

DataReceived event will fire as result of remote peer's SendData method call. When data arrives, devP2P will provide it through this event.

If you prefer to send/receive text messages, use SendText method and TextReceived event instead.

To use this event, you should implement function by yourself in the code (based on function declaration), and set devP2P.Events.DataReceived structure member to point to your function.

Code sample

C++

```
void p2p_DataReceived(devP2Plib::CP2P *p2p, int chanid, char *data, int len)
{
   printf("%s] Received %d bytes of data\r\n", p2p->MyName, len);
}

int main(int argc, char **argv)
{
/* ... */
   devP2Plib::CP2P *p1 = devP2Plib::CP2P::Create();
/* ... */
   p1->Events.DataReceived = p2p_DataReceived;
/* ... */
}
```

Platforms

Windows
Mac OSX
Linux
BSD

FileDone event

Fires when file transfer completes.

Syntax

C# | C++ | VB.NET

```
delegate void devP2P.FileDoneEvent(IntPtr Handle, int chanid, int errorno);
```

The *FileDone(Handle,chanid,error)* syntax has these parts:

Handle	Reference to the devP2P instance.
chanid	Index of the channel where file is being transferred.
error	Errors value, ErrorNone if operation is successful, or other value defining the error that occurred.

Remarks

FileDone fires when transfer completes and file is received (or sent) by the beer. *Error* argument will contain possible error if transfer failed. From now channel that was used for the transfer is freed and you can use it for some other operation.

To use this event, you should implement function by yourself in the code (based on function declaration), and set devP2P.Events.FileDone structure member to point to your function.

Code sample

C++

```
void p2p_FileDone(devP2Plib::CP2P *p2p, int chanid, devP2Plib::Errors error)
{
   printf("%s] Transfer completed with error %d %s\r\n", p2p->MyName, error, p2p->ErrorText(error));
}

int main(int argc, char **argv)
{
/* ... */
   devP2Plib::CP2P *p1 = devP2Plib::CP2P::Create();
/* ... */
   p1->Events.FileDone = p2p_FileDone;
/* ... */
}
```

Platforms

Windows
Mac OSX
Linux
BSD

FileProgress event

Fires during file transfer.

Syntax

C# | C++ | VB.NET

```
delegate void devP2P.FileProgressEvent(IntPtr Handle, int chanid, Int64 position, Int64 size);
```

The *FileProgress(Handle,chanid,position,size)* syntax has these parts:

Handle	Reference to the devP2P instance.
chanid	Index of the channel where file is being transferred.
position	Current transfer position.
size	Total size that will be transferred.

Remarks

This event fires during file transfer, both when file is being sent and received. You can monitor it's progress and check current Bandwidth if you are interested in transfer speed. When transfer finishes, FileDone will be fired.

To use this event, you should implement function by yourself in the code (based on function declaration), and set devP2P.Events.FileProgress structure member to point to your function.

Code sample

C++

```
void p2p_FileProgress(devP2Plib::CP2P *p2p, int chanid, int64 position, int64 size)
{
   printf("%s] Progress %ld/%ld (%d%%) bandwidth=%5.2f MB/sec\r\n", p2p->MyName, (long)position,
(long)size, (int)((position*100)/size), p2p->Bandwidth(chanid));
}

int main(int argc, char **argv)
{
/* ... */
   devP2Plib::CP2P *p1 = devP2Plib::CP2P::Create();
/* ... */
   p1->Events.FileProgress = p2p_FileProgress;
/* ... */
}
```

Platforms

Windows
Mac OSX
Linux
BSD

FileReceive event

Fires when remote devP2P wants to send us a file.

Type

Boolean.

Syntax

C# | C++ | VB.NET

```
delegate bool devP2P.FileReceiveEvent(IntPtr Handle, int chanid, string filename, Int64 size);
```

The *FileReceive(Handle,chanid,filename,size)* syntax has these parts:

Handle	Reference to the devP2P instance.
chanid	Index of the channel where file is being transferred.
filename	Path where file will be saved.
size	Long integer, total size of the file that will be received.
Return value	Return true if you accept file to be received. Return false to cancel the transfer.

Remarks

FileReceive event fires when remote devP2P peer wants to send us a file, and we have to decide if we will accept it or not. Event provides only filename (which you can change), and size of the file that is expected to arrive. If you return true, transfer will proceed. During the transfer, one or more FileProgress events will fire, and when transfer completes FileDone will fire.

To use this event, you should implement function by yourself in the code (based on function declaration), and set devP2P.Events.FileReceive structure member to point to your function.

Code sample

C++

```
bool p2p_FileReceive(devP2Plib::CP2P *p2p, int chanid, char *filename, int64 size)
{
   printf("%s] Receiving file %s (%ld)\r\n", p2p->MyName, filename, (long)size);
   return true;
}

int main(int argc, char **argv)
{
/* ... */
   devP2Plib::CP2P *p1 = devP2Plib::CP2P::Create();
/* ... */
   p1->Events.FileReceive = p2p_FileReceive;
/* ... */
}
```

Platforms

Windows
Mac OSX
Linux
BSD

FileSend event

Fires when local devP2P starts sending file to remote.

Type

Boolean.

Syntax

C# | C++ | VB.NET

```
delegate bool devP2P.FileSendEvent(IntPtr Handle, int chanid, string filename, Int64 size);
```

The *FileSend(Handle,chanid,filename,size)* syntax has these parts:

Handle	Reference to the devP2P instance.
chanid	Index of the channel where file is being transferred.
filename	Path to the local file.
size	Long integer, total size of the file that will be sent.
Return value	Return true if you accept file to be sent. Return false to cancel the transfer.

Remarks

FileSend event fires as result of SendFile call. If all is ok on local side, this event will fire so you can get basic idea about the transfer that will be performed. If remote side accepts it (it receives FileReceive event), one or more FileProgress events will be fired as file is being transferred. When transfer finishes, FileDone will fire.

To use this event, you should implement function by yourself in the code (based on function declaration), and set devP2P.Events.FileSend structure member to point to your function.

Code sample

C++

```
bool p2p_FileSend(devP2Plib::CP2P *p2p, int chanid, char *filename, int64 size)
{
   printf("%s] Sending file %s (%ld)\r\n", p2p->MyName, filename, (long)size);
   return true;
}

int main(int argc, char **argv)
{
/* ... */
   devP2Plib::CP2P *p1 = devP2Plib::CP2P::Create();
/* ... */
   p1->Events.FileSend = p2p_FileSend;
/* ... */
}
```

Platforms

Windows
Mac OSX
Linux
BSD

ForwardClose event

Fires when forwarding is stopped.

Syntax

C# | C++ | VB.NET

```
delegate void devP2P.ForwardCloseEvent(IntPtr Handle, int forwardid, int errorno);
```

The *ForwardClose(Handle,forwid,error)* syntax has these parts:

Handle	Reference to the devP2P instance.
forwid	Index of the forwarded channel.
error	Errors value, ErrorNone if operation is successful, or other value defining the error that occurred.

Remarks

This event is fired when specific forwarding closes. At this moment, all users that were using this channel are disconnected from devP2P.

To use this event, you should implement function by yourself in the code (based on function declaration), and set devP2P.Events.ForwardClose structure member to point to your function.

Code sample

C++

```
void p2p_ForwardClose(devP2Plib::CP2P *p2p, int forwid, devP2Plib::Errors error)
{
   printf("%s] Forward close with error %d %s\r\n", p2p->MyName, error, p2p->ErrorText(error));
}

int main(int argc, char **argv)
{
/* ... */
   devP2Plib::CP2P *p1 = devP2Plib::CP2P::Create();
/* ... */
   p1->Events.ForwardClose = p2p_ForwardClose;
/* ... */
}
```

Platforms

Windows
Mac OSX
Linux
BSD

ForwardOpen event

Fires when port forwarding is open.

Type

Boolean.

Syntax

C# | C++ | VB.NET

```
delegate bool devP2P.ForwardOpenEvent(IntPtr Handle, int forwardid, int forwardtype, string localaddress, int localport,
string remoteaddress, int remoteport);
```

The *ForwardOpen(Handle,forwid,forwardtype,localaddress,localport,remoteaddress,remoteport)* syntax has these parts:

Handle	Reference to the devP2P instance.
forwid	Forwarding channel index (0-31).
forwardtype	ForwardTypes value, type of the forward.
localaddress	Local address that is used for forwarding.
localport	Local port that is used for forwarding.
remoteaddress	Remote address that is used for forwarding.
remoteport	Remote port that is used for forwarding.
Return value	Return true if you accept the forwarding. If you return False, it will be cancelled.

Remarks

ForwardOpen event fires when either local or remote peer has requested certain port to be forwarded to remote side. For most types both localaddress/port and remoteaddress/port must be set, and forwarding is made fixed->fixed ports. In such cases all arguments have real values. For SOCKS and HTTP forwards, remoteaddress is not known since it depends on user's request, so final information about the forward will be known in UserConnected event.

You can use GetForward method to obtain information about forwarding with specified *forwid* index. Forwarding does not use any channels yet - as users are connecting, they will be taking free channels, and releasing them when they disconnect.

One forwarding can accept many users at once. For each user that connects to this forwarding, UserConnected event will fire so you can track users and deny access based on your set of rules.

To use this event, you should implement function by yourself in the code (based on function declaration), and set devP2P.Events.ForwardOpen structure member to point to your function.

Code sample

C++

```
bool p2p_ForwardOpen(devP2Plib::CP2P *p2p, int forwid, devP2Plib::ForwardTypes type, char *localaddress,
int localport, char *remoteaddress, int remoteport)
{
```

```
   const char *t = "Unknown";
   switch (type)
   {
       case devP2Plib::TCPLocalListen:    t="TCPLocalListen";break;
       case devP2Plib::TCPRemoteListen:    t="TCPRemoteListen";break;
       case devP2Plib::UDPLocalListen:    t="UDPLocalListen";break;
       case devP2Plib::UDPRemoteListen:    t="UDPRemoteListen";break;
       case devP2Plib::TCPSocksProxy:    t="TCPSocksProxy";break;
       case devP2Plib::TCPHttpProxy:        t="TCPHttpProxy";break;
   }
   printf("%s] Forward %s open from %s (%d) ==> %s (%d)\r\n", p2p->MyName, t, localaddress, localport,
remoteaddress, remoteport);
   return true;
}

int main(int argc, char **argv)
{
/* ... */
   devP2Plib::CP2P *p1 = devP2Plib::CP2P::Create();
/* ... */
   p1->Events.ForwardOpen = p2p_ForwardOpen;
/* ... */
}
```

Platforms

Windows
Mac OSX
Linux
BSD

LinkDone event

Fires when devP2P links with remote devP2P instance.

Syntax

C# | C++ | VB.NET

```
delegate void devP2P.LinkDoneEvent(IntPtr Handle, string address, int port, int errorno);
```

The *LinkDone(Handle,address,port)* syntax has these parts:

Handle	Reference to the devP2P instance.
address	IP address of remote peer.
port	Port of remote peer.

Remarks

After remote devP2P peer is located (through Search method), and Link method was called, devP2P will try to connect to remote devP2P peer. After successful link, this event will be fired. Now you can call, for example, StartForward to begin forwarding ports between local and remote side, can send text messages etc.

Note that this event can be fired if you called Start method, and doing nothing, if remote peer initiated Link method call to you, from his side.

To obtain connection type (UDP/TCP), you can use ConnectionType property.

To use this event, you should implement function by yourself in the code (based on function declaration), and set devP2P.Events.LinkDone structure member to point to your function.

Code sample

C++

```
void p2p_LinkDone(devP2Plib::CP2P *p2p, char *address, int port, devP2Plib::Errors error)
{
   if (!error)
       printf("%s] Connected to %s (%d)\r\n", p2p->MyName, address, port);
   else
       printf("%s] Connection failed with error %s (%d)\r\n", p2p->MyName, p2p->ErrorText(error), error);
}


int main(int argc, char **argv)
{
/* ... */
   devP2Plib::CP2P *p1 = devP2Plib::CP2P::Create();
/* ... */
   p1->Events.LinkDone = p2p_LinkDone;
/* ... */
}
```

Platforms

Windows

Mac OSX
Linux
BSD

NewUPNPMapping event

Fires when new UPnP port mapping was created.

Syntax

C# | C++ | VB.NET

```
delegate void devP2P.NewUPNPMappingEvent(IntPtr Handle, string ExtAddress, string IntAddress, int TCPPort, int UDPPort);
```

The *NewUPNPMapping(Handle,ExtAddress,IntAddress,TCPPort,UDPPort)* syntax has these parts:

Handle	Reference to the devP2P instance.
ExtAddress	IP address on UPNP device.
IntAddress	Our IP address.
TCPPort	TCP port on the UPNP device that forwards back to us.
UDPPort	UDP port on the UPNP device that forwards back to us.

Remarks

NewUPnPMapping event is fired when devP2P maps external port on UPnP device (usually firewall or router device) so that remote peer can connect to that device to establish connection with your devP2P instance. You do not need to make any special setup on your that device - as long as UPnP service is provided, devP2P will know how to use it.

You should, however, call UPNPInit in your code to enable UPnP.

To use this event, you should implement function by yourself in the code (based on function declaration), and set devP2P.Events.NewUPNPMapping structure member to point to your function.

Code sample

C++

```
void p2p_NewUPNPMapping(devP2Plib::CP2P *p2p, char *ExtAddress, char *IntAddress, int TCPPort, int
UDPPort)
{
   if (TCPPort) printf("%s] New TCP mapping: %s->%s (%d)\r\n", p2p->MyName, ExtAddress, IntAddress,
TCPPort);
   if (UDPPort) printf("%s] New UDP mapping: %s->%s (%d)\r\n", p2p->MyName, ExtAddress, IntAddress,
UDPPort);
}

int main(int argc, char **argv)
{
/* ... */
   devP2Plib::CP2P *p1 = devP2Plib::CP2P::Create();
/* ... */
   p1->Events.NewUPNPMapping = p2p_NewUPNPMapping;
/* ... */
}
```

Platforms

Windows
Mac OSX
Linux
BSD

Ping event

Fires when Ping packet comes from remote peer.

Syntax

C# | C++ | VB.NET

```
delegate void devP2P.PingEvent(IntPtr Handle);
```

The *Ping(Handle)* syntax has these parts:

Handle	Reference to the devP2P instance.

Remarks

Ping event can be fired by auto timeout feature, which sends internal PING packets each few seconds to determine if link is still valid, when no data is transferred during that time. You can also call Ping method by yourself to initiate this check.

Platforms

Windows
Mac OSX
Linux
BSD

SearchDone event

Fires when Search method completes its search for remote devP2P peer.

Syntax

C# | C++ | VB.NET

```
delegate void devP2P.SearchDoneEvent(IntPtr Handle, string peerName, string peerXML, string customData, int errorno);
```

The *SearchDone(Handle,peerName,peerXML,customData,error)* syntax has these parts:

Handle	Reference to the devP2P instance.
peerName	Name of the peer you are connected to.
peerXML	XML containing list of IP address where remote listens.
customData	Custom data provided by remote in Search method.
error	Errors value, *ErrorNone* if operation is successful, or other value defining the error that occurred.

Remarks

SearchDone event is fired after Search method was called, and devP2P finishes it search for remote peer. At this time this can be successful search (*ErrorCode* set to ErrorNone), or some error may have been occurred.

Even search was successful, it doesn't mean you're already linked to other devP2P. At this point you have enough information to attempt to connect using Link method - and you should do so within next few seconds otherwise IP/Port may become invalid (since your router can release his NAT translation table). It is possible that **peerName** argument is different than PeerName property, in case you used wildcards in PeerName property.

Typically, you will have basic code inside this event, such as

```
devP2P->Link(peeraddress);
```

but you MUST have code similar to above in your SearchDone method. Default implementation of SearchDone does call Link method, but if you change it you should manually call Link.
You can also inspect value of *peeraddress* argument, and add or remove elements from it.

To use this event, you should implement function by yourself in the code (based on function declaration), and set devP2P.Events.SearchDone structure member to point to your function.

Code sample

C++

```
void p2p_SearchDone(devP2Plib::CP2P *p2p, char *peerName, char *peerXML, char *customData,
devP2Plib::Errors error)
{
   printf("%s] Search finished with error %d %s\r\n", peerName, error, p2p->ErrorText(error));
   if (!error)
       p2p->Link(peerAddress);
}

int main(int argc, char **argv)
{
/* ... */
```

```
   devP2Plib::CP2P *p1 = devP2Plib::CP2P::Create();
/* ... */
   p1->Events.SearchDone = p2p_SearchDone;
/* ... */
}
```

Platforms

Windows
Mac OSX
Linux
BSD

SearchStart event

Fires when search has started.

Syntax

C# | C++ | VB.NET

```
delegate void devP2P.SearchStartEvent(IntPtr Handle, int binds);
```

The *SearchStart(Handle,binds)* syntax has these parts:

Handle	Reference to the devP2P instance.
binds	Custom bind parameters passed to remote peer.

Remarks

This event is fired after you call Start method, allowing you to specify custom bindip parameters - list of local IP/Port points where user can try to connect. It will already contain a list like this:

```
T192.168.1.1 1024
U192.168.1.1 23234
T1.2.3.4 2344
U1.2.3.4 32532
```

etc.. It is a list of local interfaces where devP2P listens for remote peer's connection. If UPNPinit was called, and UPNP is available on your system, this list can also contain external IP address on your router where devP2P managed to open a port that will forward back to it. However, if you're aware that some other IP/Port also points to your devP2P instance, you can add it here, line by line - starting with 'T' for TCP, 'U' for UDP, and then following with IP, one space, port, and newline.

To use this event, you should implement function by yourself in the code (based on function declaration), and set devP2P.Events.SearchStart structure member to point to your function.

Code sample

C++

```
void p2p_SearchStart(devP2Plib::CP2P *p2p, char *binds)
{
   printf("%s] started searching for %s at %s (%d)\r\n", p2p->MyName, p2p->PeerName, p2p-
>MediatorAddress, p2p->MediatorPort);
}

int main(int argc, char **argv)
{
/* ... */
   devP2Plib::CP2P *p1 = devP2Plib::CP2P::Create();
/* ... */
   p1->Events.SearchStart = p2p_SearchStart;
/* ... */
}
```

Platforms

Windows
Mac OSX
Linux
BSD

StateChange event

Fires when devP2P changes its state.

Syntax

C# | C++ | VB.NET

```
delegate void devP2P.StateChangeEvent(IntPtr Handle, int state);
```

The *StateChange(Handle,state)* syntax has these parts:

Handle	Reference to the devP2P instance.
state	States value. Current state.

Remarks

StateChange event is fired each time devP2P changes its internal state. Initially it is set to *StateDisconnected*, but as you call devP2P's methods it will be changed to *StateListening*, *StateLinking* etc..

Using this event you can determine whether there is anything happening with devP2P. You can access current state using State property.

To use this event, you should implement function by yourself in the code (based on function declaration), and set devP2P.Events.StateChange structure member to point to your function.

Code sample

C++

```
void p2p_StateChange(devP2Plib::CP2P *p2p, devP2Plib::States state)
{
   printf("%s] State changed to %s\r\n", p2p->MyName, p2p->StateText(state));
}

int main(int argc, char **argv)
{
/* ... */
   devP2Plib::CP2P *p1 = devP2Plib::CP2P::Create();
/* ... */
   p1->Events.StateChange = p2p_StateChange;
/* ... */
}
```

Platforms

Windows
Mac OSX
Linux
BSD

Stopped event

Fires when devP2P stops working and goes offline.

Syntax

C# | C++ | VB.NET

```
delegate void devP2P.StoppedEvent(IntPtr Handle, int errorno);
```

The *Stopped(Handle,error)* syntax has these parts:

Handle	Reference to the devP2P instance.
error	Errors value, ErrorNone if operation is successful, or other value defining the error that occurred.

Remarks

This event fires when devP2P stops it's operation and goes offline, no matter if it was previously Connected to remote peer or not. If it stopped due to an error, *Error* argument will provide info about the error. If it was result of your call to Stop method, *Error* will be empty. You should now call Start method again to accept new connections.

When devP2P successfully connects to remote peer, you cannot reuse same devP2P instance, so there is no *Disconnect* method in devP2P. If you decide you want to reconnect, you must use new devP2P instance (or Stop/Start current one) so internal buffers are cleared. Otherwise, devP2P could be left in unknown state - dealing with mediator, local UDP/TCP listening sockets etc..

To use this event, you should implement function by yourself in the code (based on function declaration), and set devP2P.Events.Stopped structure member to point to your function.

Code sample

C++

```
void p2p_Stopped(devP2Plib::CP2P *p2p, devP2Plib::Errors error)
{
   printf("%s] Stopped, error %s\r\n", p2p->MyName, p2p->ErrorText(error));
}

int main(int argc, char **argv)
{
/* ... */
   devP2Plib::CP2P *p1 = devP2Plib::CP2P::Create();
/* ... */
   p1->Events.Stopped = p2p_Stopped;
/* ... */
}
```

Platforms

Windows
Mac OSX
Linux
BSD

TextReceived event

Fires when text message arrives from remote side.

Syntax

C# | C++ | VB.NET

```
delegate void devP2P.TextReceivedEvent(IntPtr Handle, int chanid, string text);
```

The *TextReceived(Handle,chanid,text)* syntax has these parts:

Handle	Reference to the devP2P instance.
chanid	Index of the channel where data arrived.
text	String that was received from remote peer.

Remarks

TextReceived event is fired when remote side uses SendText method to send us short text message. On local side you can show this text to your application, or you can use it to send/receive various short commands between devP2P peers, so that you can make certain custom actions.

If you want to send non-text (binary) data, use SendData method instead.

To use this event, you should implement function by yourself in the code (based on function declaration), and set devP2P.Events.TextReceived structure member to point to your function.

Platforms

Windows
Mac OSX
Linux
BSD

UserConnected event

Fires when user connects to forwarding channel.

Type

Boolean.

Syntax

C# | C++ | VB.NET

```
delegate bool devP2P.UserConnectedEvent(IntPtr Handle, int forwardid, int chanid, IntPtr userhandle);
```

The *UserConnected(Handle,forwid,chanid,user)* syntax has these parts:

Handle	Reference to the devP2P instance.
forwid	Index of the forwarded channel.
chanid	Index of the channel that user will use.
user	Reference to CP2PForwardUser with user details.
Return value	Return true if you accept the user, or False to disconnect him.

Remarks

UserConnected event is fired after you have allowed user to connect and use forwarding they have chosen. From this point, user will be communicating with remote service through encrypted channel.
There will be no further interference between devP2P and the user - all his data will be forwarded to remote devP2P peer, and vice versa. When user disconnects, UserDisconnected event will be fired.

To use this event, you should implement function by yourself in the code (based on function declaration), and set devP2P.Events.UserConnected structure member to point to your function.

Code sample

C++

```
bool p2p_UserConnected(devP2Plib::CP2P *p2p, int forwid, int chanid, devP2Plib::CP2PForwardUser *user)
{
   printf("%s] User connected\r\n", p2p->MyName);
   return true;
}

int main(int argc, char **argv)
{
/* ... */
   devP2Plib::CP2P *p1 = devP2Plib::CP2P::Create();
/* ... */
   p1->Events.UserConnected = p2p_UserConnected;
/* ... */
}
```

Platforms

Windows
Mac OSX
Linux
BSD

UserDisconnected event

Fires when user disconnects from the forwarded channel.

Syntax

C# | C++ | VB.NET

```
delegate void devP2P.UserDisconnectedEvent(IntPtr Handle, int forwardid, int chanid, IntPtr userhandle, int errorno);
```

The *UserDisconnected(Handle,forwid,chanid,user,error)* syntax has these parts:

Handle	Reference to the devP2P instance.
forwid	Index of the forwarded channel.
chanid	Index of the channel that was used by this user.
user	Reference to CP2PForwardUser with user details.
error	Errors value, possible error that caused user to disconnect.

Remarks

This event is fired when user leaves the channel. At this point he will be removed from the collection of all connected users, so this is last change to access information about the user. If disconnection was result of an error, *Error* argument will contain description of the error.

Note that if UDP channel was used, UserDisconnected event may be result of user being idle for certain amount of seconds.

To use this event, you should implement function by yourself in the code (based on function declaration), and set devP2P.Events.UserDisconnected structure member to point to your function.

Code sample

C++

```
void p2p_UserDisconnected(devP2Plib::CP2P *p2p, int forwid, int chanid, devP2Plib::CP2PForwardUser *user,
devP2Plib::Errors error)
{
   printf("%s] User disconnected with error %d %s\r\n", p2p->MyName, error, p2p->ErrorText(error));
}

int main(int argc, char **argv)
{
/* ... */
   devP2Plib::CP2P *p1 = devP2Plib::CP2P::Create();
/* ... */
   p1->Events.UserDisconnected = p2p_UserDisconnected;
/* ... */
}
```

Platforms

Windows
Mac OSX

Linux
BSD

devVPN

Provides VPN through dynamic array of devP2P instances.

Create	Creates new devVPN class instance.
Destroy	Destroys current devVPN instance.

Properties

Events	Reference to event handlers.
MediatorAddress	Holds IP address (or hostname) of the mediator.
MediatorPort	Holds port of the mediator.
MyName	Holds user defined identity ID of local devVPN peer.
State	Returns current devVPN state.
Tag	Tag for misc usage.

Methods

ErrorText	Returns text representation of the error.
Search	Starts searching for other peers.
SetAdapter	Sets adapter to be used with devVPN.
Start	Starts devVPN, and binds local interfaces.
StateText	Returns text representation of the state.
Stop	Stops routing network packets, and disconnects all peers.

Events

PeerConnected	Fires when new peer is successfully connected.
PeerConnecting	Fires when remote peer wants to connect.
PeerDisconnected	Fires when peer is disconnected from devVPN.
StateChange	Fires when devVPN changes its state.

Platforms

Windows
Mac OSX
Linux
BSD
iPhone IOS

Create method

Creates new devVPN class instance.

Syntax

C# | C++ | VB.NET

```
IntPtr devVPN.Create();
```

The *Create()* syntax has these parts:

Return value	Reference to new created CVPN instance.

Remarks

This is a static method that creates new instance of devVPN. After instance is successfully obtained and used, you should destroy it using Destroy method.

You should not delete the instance by yourself. Always use Destroy method instead.

Make sure you initialized the library first, by calling libInit function!

Code sample

C++

```
// Initialize devP2P library
devP2Plib::libInit();
devP2Plib::upnpInit();
// Give some time for UPNP to exchange packets
Sleep(200);

devP2Plib::CVPN *v1 = devP2Plib::CVPN::Create();
// ... //
```

Platforms

Windows
Mac OSX
Linux
BSD
iPhone IOS

Destroy

Destroys current devVPN instance.

Syntax

C# | C++ | VB.NET

```
void devVPN.Destroy(IntPtr Handle);
```

Remarks

This method destroys the devVPN instance, obtained by Create method. When it's not used anymore, it is internally deleted.

Properties

Events	Reference to event handlers.
MediatorAddress	Holds IP address (or hostname) of the mediator.
MediatorPort	Holds port of the mediator.
MyName	Holds user defined identity ID of local devVPN peer.
State	Returns current devVPN state.
Tag	Tag for misc usage.

Platforms

Windows
Mac OSX
Linux
BSD
iPhone IOS

Events property

Reference to event handlers.

Type

VPNEventsStruct structure.

Syntax

C++

```
VPNEventsStruct Events;
```

The *Events()* syntax has these parts:

Return value	Returns pointer to internal events structure.

Remarks

To use specific event with devVPN, you must implement your own function that has same declaration as the event, and give a reference to VPNEventsStruct for the function. VPNEventStruct members correspond to events, and default to NULL. Below in code samples is shown how to do it for some events.

This is the declaration of VPNEventsStruct

```
typedef struct VPNEventsStruct
{
    void (*StateChange)(CVPN *vpn, States state);
    bool (*PeerConnecting)(CVPN *vpn, CP2P *p2p, char *peerName, char *peerAddrs);
    void (*PeerConnected)(CVPN *vpn, CP2P *p2p, char *address, int port);
    void (*PeerDisconnected)(CVPN *vpn, CP2P *p2p, Errors error);
} VPNEventsStruct;
```

Code sample

C++

```
void vpn_StateChange(devP2Plib::CVPN *vpn, devP2Plib::States state)
{
   printf("State changed to %s\r\n", vpn->StateText(state));
}

bool vpn_PeerConnecting(devP2Plib::CVPN *vpn, devP2Plib::CP2P *p2p, char *peerName, char *peerAddrs)
{
   printf("PeerConnecting %s\r\n", peerName);
   return true;
}

void vpn_PeerConnected(devP2Plib::CVPN *vpn, devP2Plib::CP2P *p2p, char *address, int port)
{
   struct in_addr ina;
   ina.s_addr = p2p->PeerAdapterIP;
   printf("PeerConnected %s, IP %s\r\n", p2p->PeerName, inet_ntoa(ina));
}
```

```
void vpn_PeerDisconnected(devP2Plib::CVPN *vpn, devP2Plib::CP2P *p2p, devP2Plib::Errors error)
{
   struct in_addr ina;
   ina.s_addr = p2p->PeerAdapterIP;
   printf("PeerDisconnected %s, IP %s (%s)\r\n", p2p->PeerName, inet_ntoa(ina), vpn->ErrorText(error));
}

/* ..... */

int main(int argc, char **argv)
{
   devP2Plib::CVPN *vpnhost;

   vpnhost = devP2Plib::CVPN::Create();
   vpnhost->Events.PeerConnected = vpn_PeerConnected;
   vpnhost->Events.PeerConnecting = vpn_PeerConnecting;
   vpnhost->Events.PeerDisconnected = vpn_PeerDisconnected;
   vpnhost->Events.StateChange = vpn_StateChange;
/* ..... */
}
```

Platforms

Windows
Mac OSX
Linux
BSD
iPhone IOS

MediatorAddress property

Holds IP address (or hostname) of the mediator.

Type

String.

Syntax

C# | C++ | VB.NET

```
void devVPN.GetMediatorAddress(IntPtr Handle, StringBuilder buffer);
```

The *MediatorAddress(Handle,buffer)* syntax has these parts:

Handle	Reference to the devVPN instance.
buffer	Buffer where output is stored.

```
void devVPN.SetMediatorAddress(IntPtr Handle, string Value);
```

The *MediatorAddress(Handle,value)* syntax has these parts:

Handle	Reference to the devVPN instance.
value	New mediator address to set.

Remarks

Set this property to hostname of mediator that is used with Search method, together with MediatorPort property. devVPN will send mediator requests to that IP/Port to locate and possibly request connection with remote peer.

UDP connection is used for mediator, to help with UDP hole punching for direct peer-to-peer connection. Even if you use only TCP protocol in Start method, mediator can be used to obtain information about announced TCP ports open by remote peer.

Note that devVPN will remember this property through address reference, so keep your buffers static and valid as long as devVPN needs it. devVPN will not free or touch allocated memory in any way.

Platforms

Windows
Mac OSX
Linux
BSD
iPhone IOS

MediatorPort property

Holds port of the mediator.

Type

Integer.

Syntax

C# | C++ | VB.NET

```
void devVPN.SetMediatorPort(IntPtr Handle, int Value);
```

The *MediatorPort(Handle,value)* syntax has these parts:

Handle	Reference to the devVPN instance.
value	Port where mediator listens.

```
int devVPN.GetMediatorPort(IntPtr Handle);
```

The *MediatorPort(Handle)* syntax has these parts:

Handle	Reference to the devVPN instance.
Return value	Integer value representing mediator's port.

Remarks

Set this property to port where mediator listens for Search method, together with MediatorAddress property. devVPN will send mediator requests to that IP/Port to locate and possibly request connection with remote peer.

UDP connection is used for mediator, to help with UDP hole punching for direct peer-to-peer connection. Even if you use only TCP protocol in Start method, mediator can be used to obtain information about announced TCP ports open by remote peer.

Platforms

Windows
Mac OSX
Linux
BSD
iPhone IOS

MyName property

Holds user defined identity ID of local devVPN peer.

Type

String.

Platforms

Windows
Mac OSX
Linux
BSD
iPhone IOS

State property

Returns current devVPN state.

Type

States enumeration.

Syntax

C# | C++ | VB.NET

```
int devVPN.GetState(IntPtr Handle);
```

The *State(Handle)* syntax has these parts:

Handle	Reference to the devVPN instance.
Return value	Current state from States enumeration.

Remarks

State property returns current devVPN state. If devVPN is completely idle, State will hold **StateStopped** value. As soon as devVPN starts with some activity, StateChange event will fire where you can keep track on devVPN's behavior.

Platforms

Windows
Mac OSX
Linux
BSD
iPhone IOS

Tag property

Tag for misc usage.

Type

Object.

Syntax

C# | C++ | VB.NET

```
void devVPN.SetTag(IntPtr Handle, object Value);
```

The *Tag(Handle,value)* syntax has these parts:

Handle	Reference to the devVPN instance.
value	Object that holds your data.

```
object devVPN.GetTag(IntPtr Handle);
```

The *Tag(Handle)* syntax has these parts:

Handle	Reference to the devVPN instance.
Return value	Object that holds your data.

Remarks

You can use this property to store pointer to your own custom data that will be kept by devVPN. devVPN will not interfere with this value in any way (it will not, for example, try to free that memory).

Platforms

Windows
Mac OSX
Linux
BSD
iPhone IOS

Methods

ErrorText	Returns text representation of the error.
Search	Starts searching for other peers.
SetAdapter	Sets adapter to be used with devVPN.
Start	Starts devVPN, and binds local interfaces.
StateText	Returns text representation of the state.
Stop	Stops routing network packets, and disconnects all peers.

Platforms

Windows
Mac OSX
Linux
BSD
iPhone IOS

ErrorText method

Returns text representation of the error.

Type

String.

Syntax

C# | C++ | VB.NET

```
void devVPN.ErrorText(IntPtr Handle, int errorno, StringBuilder buffer);
```

The *ErrorText(Handle,error,buffer)* syntax has these parts:

Handle	Reference to the devVPN instance.
error	Errors value.
buffer	String buffer where error text will be stored.

Remarks

This method will return text representation of the error, in English. You can use it to show user-friendly information in your application.

Platforms

Windows
Mac OSX
Linux
BSD
iPhone IOS

Search method

Starts searching for other peers.

Type

Integer.

Syntax

C++

```
int Search(char *peerNames, char *password, bool forever);
```

The *Search(peerNames,password,forever)* syntax has these parts:

peerNames	String. (Wildcard) list of accepted peer names that can connect.
password	String. Password to encrypt data with remote peer.
forever	Boolean. When set to True, search for other peers will not end after first match is found.
Return value	0 (ErrorNone) if success, otherwise negative value from Errors enumeration.

Remarks

Search method should be called after the Start method, to initiate searching for remote peers. You can search for specific peer, or for more than one peer, by setting peerNames to contain '*' at the end of the allowed names. This way you can create one-to-many VPN connection, and devVPN will create devP2P instances on the fly as new peers arrive.

If **forever** is set to True, devVPN will continue searching for new peers and added them to local collection as they arrive. This is typical usage for server kind of applications. If you're creating client side, it is suggested to set **forever** to False, and possibly if connection is broken after some time re-initiate searching again.

You can call Search method anytime you wish, to start your search for different peers. When doing so, previous Search will be cancelled, and new one will be invoked.

Platforms

Windows
Mac OSX
Linux
BSD
iPhone IOS

SetAdapter method

Sets adapter to be used with devVPN.

Type

Integer.

Syntax

C++

```
int SetAdapter(CVPNInterface *adapter);
```

The *SetAdapter(adapter)* syntax has these parts:

adapter	Reference to CVPNinterface object, returned by VPNAdapter function.
Return value	0 (ErrorNone) if success, otherwise negative value from Errors enumeration.

Remarks

Before Start method is called, you must use SetAdapter to choose which adapter will be used to forward network traffic between your and remote peers. It must have valid IP and Netmask, and all peers will also need to have IP address in same network range (defined by Netmask) in order to successfully route network packets.

Platforms

Windows
Mac OSX
Linux
BSD
iPhone IOS

Start method

Starts devVPN, and binds local interfaces.

Type

Integer.

Remarks

Start method will bind local ports, open local network interface, and prepare itself to accept new peers. In order to actually locate remote peers, you should call Search method after calling the Start.

Platforms

Windows
Mac OSX
Linux
BSD
iPhone IOS

StateText method

Returns text representation of the state.

Type

String.

Syntax

C++

```
static const char *StateText(States state);
```

The *StateText(state)* syntax has these parts:

state	States value.
Return value	Text representation of the state.

Remarks

This method will return text representation of the state, in English. You can use it to show user-friendly information in your application. Usually you will use this method in StateChange event.

Platforms

Windows
Mac OSX
Linux
BSD
iPhone IOS

Stop method

Stops routing network packets, and disconnects all peers.

Syntax

C++

```
void Stop(void);
```

Remarks

This methos will disconnect all peers and stop routing network packets. It will also free adapter reference, so you can make changes on the adapter. To start devVPN again, use Start method.

Platforms

Windows
Mac OSX
Linux
BSD
iPhone IOS

Events

PeerConnected	Fires when new peer is successfully connected.
PeerConnecting	Fires when remote peer wants to connect.
PeerDisconnected	Fires when peer is disconnected from devVPN.
StateChange	Fires when devVPN changes its state.

Platforms

Windows
Mac OSX
Linux
BSD
iPhone IOS

PeerConnected event

Fires when new peer is successfully connected.

Syntax

C++

```
void PeerConnected(CVPN *vpn, CP2P *p2p, char *address, int port);
```

The *PeerConnected(vpn,p2p,address,port)* syntax has these parts:

vpn	Pointer to devVPN instance that fired the event.
p2p	Pointer to devP2P instance that is connected.
address	String that contains real IP address of the peer.
port	Integer that contains port of the peer.

Remarks

This event is fired after connection with the peer is successfully established. You can now access the peer through his virtual IP address.

To use this event, you should implement function by yourself in the code (based on function declaration), and set devVPN.Events.PeerConnected structure member to point to your function.

Platforms

Windows
Mac OSX
Linux
BSD
iPhone IOS

PeerConnecting event

Fires when remote peer wants to connect.

Type

Boolean value.

Syntax

C++

```
bool PeerConnecting(CVPN *vpn, CP2P *p2p, char *peerName, char *peerAddrs);
```

The *PeerConnecting(vpn,p2p,peerName,peerAddrs)* syntax has these parts:

vpn	Pointer to devVPN instance that fired the event.
p2p	Pointer to devP2P instance that wants to connect.
peerName	Name of the peer that wants to connect.
peerAddrs	Virtual IP address of the peer that you can use to access the peer.
Return value	Return True if you accept the connection with the peer.

Remarks

After Search method is called, and mediator finds a match, this event will be fired so you can decide if you want to connect with this peer. If you want to connect, return **True**, and connection will be accepted.

After connection is (successfully) established, PeerConnected event will be fired.

To use this event, you should implement function by yourself in the code (based on function declaration), and set devVPN.Events.PeerConnecting structure member to point to your function.

Platforms

Windows
Mac OSX
Linux
BSD
iPhone IOS

PeerDisconnected event

Fires when peer is disconnected from devVPN.

Remarks

This event fires after connection with specific peer was broken. Peer can be disconnected by your intention, or due to error that has occurred. If error occurred, error argument will contain description of the error.

To use this event, you should implement function by yourself in the code (based on function declaration), and set devVPN.Events.PeerDisconnected structure member to point to your function.

Platforms

Windows
Mac OSX
Linux
BSD
iPhone IOS

StateChange event

Fires when devVPN changes its state.

Remarks

StateChange event is fired each time devVPN changes its internal state. Initially it is set to *StateDisconnected*, but as you call devVPN's methods it will be changed to *StateListening*, *StateSearching* etc..

Using this event you can determine whether there is anything happening with devVPN. You can access current state using State property.

To use this event, you should implement function by yourself in the code (based on function declaration), and set devVPN.Events.StateChange structure member to point to your function.

Platforms

Windows
Mac OSX
Linux
BSD
iPhone IOS

Objects

CP2PForwardUser	Holds information about user connected to forwarded channel.
CP2PForward	Holds information about forwarding channel.
CVPNInterface	Holds information about virtual network adapter.

Platforms

Windows
Mac OSX
Linux
BSD

CP2PForwardUser

Holds information about user connected to forwarded channel.

Disconnect	Disconnects user from the devP2P forwarding.
Tag	Tag for misc usage.

Remarks

Reference to this object is provided in UserConnected and UserDisconnected events, as users are connecting and using your forwarded channel.

Platforms

Windows
Mac OSX
Linux
BSD

Disconnect method

Disconnects user from the devP2P forwarding.

Syntax

C# | C++ | VB.NET

```
void devP2P.ForwardUser_Disconnect(IntPtr UserHandle);
```

The *Disconnect(UserHandle)* syntax has these parts:

UserHandle	Reference to the CP2PForwardUser instance.

Platforms

Windows
Mac OSX
Linux
BSD

Tag property

Tag for misc usage.

Type

Object.

Syntax

C# | C++ | VB.NET

```
void devP2P.ForwardUser_SetTag(IntPtr Handle, object Value);
```

The *Tag(UserHandle,Value)* syntax has these parts:

UserHandle	Reference to the CP2PForwardUser instance.
Value	Your object with custom data.

```
object devP2P.ForwardUser_GetTag(IntPtr Handle);
```

The *Tag(UserHandle)* syntax has these parts:

UserHandle	Reference to the CP2PForwardUser instance.
Return value	Your object with custom data.

Remarks

Custom data that is kept by this user instance.

Platforms

Windows
Mac OSX
Linux
BSD

CP2PForward

Holds information about forwarding channel.

DisconnectUser	Disconnect specific user from the forwarding.
ForwardType	Determines type of the forwarding used by connected user.
IsOpen	Determines if forwarding is open and accepting connections.
LocalAddress	Holds local IP address of connected user.
LocalPort	Holds local port of connected user.
RemoteAddress	Holds remote IP address of connected user.
RemotePort	Holds remote port of connected user.
Tag	Tag for misc usage.

Remarks

CP2PForwarding object is used with ForwardOpen, GetForward and similar functions that deal with port forwarding.

Platforms

Windows
Mac OSX
Linux
BSD

DisconnectUser method

Disconnect specific user from the forwarding.

Syntax

C# | C++ | VB.NET

```
void devP2P.Forward_DisconnectUser(IntPtr ForwardHandle, IntPtr UserHandle);
```

The *DisconnectUser(ForwardHandle,UserHandle)* syntax has these parts:

ForwardHandle	Reference to the CP2PForward instance.
UserHandle	Reference to CP2PForwardUser that will be disconnected.

Platforms

Windows
Mac OSX
Linux
BSD

ForwardType property

Determines type of the forwarding used by connected user.

Type

ForwardTypes value.

Syntax

C# | C++ | VB.NET

```
void devP2P.Forward_SetForwardType(IntPtr ForwardHandle, int value);
```

The *ForwardType(ForwardHandle)* syntax has these parts:

ForwardHandle	Reference to the CP2PForward instance.

```
int devP2P.Forward_GetForwardType(IntPtr ForwardHandle);
```

The *ForwardType(ForwardHandle)* syntax has these parts:

ForwardHandle	Reference to the CP2PForward instance.

Platforms

Windows
Mac OSX
Linux
BSD

IsOpen method

Determines if forwarding is open and accepting connections.

Type

Boolean.

Syntax

C# | C++ | VB.NET

```
bool devP2P.Forward_IsOpen(IntPtr ForwardHandle);
```

The *IsOpen(ForwardHandle)* syntax has these parts:

ForwardHandle	Reference to the CP2PForward instance.

Platforms

Windows
Mac OSX
Linux
BSD

LocalAddress property

Holds local IP address of connected user.

Type

String.

Syntax

C# | C++ | VB.NET

```
void devP2P.Forward_SetLocalAddress(IntPtr ForwardHandle, string value);
```

The *LocalAddress(ForwardHandle,value)* syntax has these parts:

ForwardHandle	Reference to the CP2PForward instance.
value	String buffer with new local address.

```
void devP2P.Forward_GetLocalAddress(IntPtr ForwardHandle, StringBuilder buffer);
```

The *LocalAddress(ForwardHandle)* syntax has these parts:

ForwardHandle	Reference to the CP2PForward instance.

Platforms

Windows
Mac OSX
Linux
BSD

LocalPort property

Holds local port of connected user.

Type

Integer.

Syntax

C# | C++ | VB.NET

```
void devP2P.Forward_SetLocalPort(IntPtr ForwardHandle, int value);
```

The *LocalPort(ForwardHandle)* syntax has these parts:

ForwardHandle	Reference to the CP2PForward instance.

```
int devP2P.Forward_GetLocalPort(IntPtr ForwardHandle);
```

The *LocalPort(ForwardHandle)* syntax has these parts:

ForwardHandle	Reference to the CP2PForward instance.

Platforms

Windows
Mac OSX
Linux
BSD

RemoteAddress

Holds remote IP address of connected user.

Type

String.

Syntax

C# | C++ | VB.NET

```
void devP2P.Forward_SetRemoteAddress(IntPtr ForwardHandle, string value);
```

The *RemoteAddress(ForwardHandle,value)* syntax has these parts:

ForwardHandle	Reference to the CP2PForward instance.
value	String buffer with new remote address.

```
void devP2P.Forward_GetRemoteAddress(IntPtr ForwardHandle, StringBuilder buffer);
```

The *RemoteAddress(ForwardHandle)* syntax has these parts:

ForwardHandle	Reference to the CP2PForward instance.

Platforms

Windows
Mac OSX
Linux
BSD

RemotePort property

Holds remote port of connected user.

Type

Integer.

Syntax

C# | C++ | VB.NET

```
int devP2P.Forward_GetRemotePort(IntPtr ForwardHandle);
```

The *RemotePort(ForwardHandle)* syntax has these parts:

ForwardHandle	Reference to the CP2PForward instance.

Platforms

Windows
Mac OSX
Linux
BSD

Tag property

Tag for misc usage.

Type

Object.

Syntax

C# | C++

```
void devP2P.Forward_SetTag(IntPtr Handle, object Value);
```

The *Tag(ForwardHandle)* syntax has these parts:

ForwardHandle	Reference to the CP2PForward instance.

```
object devP2P.Forward_GetTag(IntPtr Handle);
```

The *Tag(ForwardHandle)* syntax has these parts:

ForwardHandle	Reference to the CP2PForward instance.

Remarks

Custom data that is kept by this forward instance.

Platforms

Windows
Mac OSX
Linux
BSD

CVPNInterface

Holds information about virtual network adapter.

Guid	Returns GUID of the interface.
LocalIP	Holds local IP address of the interface.
LocalMAC	Holds MAC address of local interface.
LocalNetmask	Holds netmask of local interface.
Name	Returns name of the interface.
SetIP	Attempts to set local IP and netmask for the adapter.

Remarks

CVPNInterface object holds information about each Interface (or network adapter) found on your computer that is capable of being used by devP2P. In order to use any of those adapters you should set its IP address and Netmask, and put local peer and remote peer to same network. For example, you can set local side to 192.168.1.1 (netmask 255.255.255.0) and remote peer's IP address to 192.168.1.2 (netmask 255.255.255.0). If you use wrong IP/Netmask combination, it is possible that sockets will not route packets correctly to remote side.

devP2P collects all raw packets from VPN adapter and routes it to remote side. Remote side then unpacks it and "pushes" to adapter so it becomes available to sockets stack. System do not see a difference between real network adapters which are connected through cable, and our virtual network adapters which are connected through devP2P.

Currently Windows version of devP2P supports included 'WeOnlyDo Network Adapter', but it can also use Wippien's adapter, and OpenVPN's adapter. Only one process at a time can use one adapter, but multiple instances of devP2P in same process (inside your application) can all share same adapter.

Platforms

Windows
Mac OSX
Linux
BSD

Guid property

Returns GUID of the interface.

Type

String.

Syntax

C++

```
char Guid[1024];
```

Remarks

Guid holds unique GUID Windows assigned for the adapter. You can use this value to access the adapter directly, or to locate it in the registry to review his properties etc.
Changing this value does not change GUID for the adapter, it only affects how devP2P sees it. Unless you know what you're doing you should not change this value.

Platforms

Windows
Mac OSX
Linux
BSD

LocalIP property

Holds local IP address of the interface.

Type

Unsigned integer.

Syntax

C++

```
unsigned int LocalIP;
```

Remarks

LocalIP property holds IP address of the adapter, if set. If DHCP is enabled on the adapter then LocalIP will have value 0. You must set local IP address of the adapter manually (through right-click on the adapter, and selecting Properties) before devP2P can use it correctly. **Setting new value in LocalIP property *does not* change real IP address of the adapter.**

When setting IP address of the adapter, make sure you also set its correct Netmask. IP/Netmask combination on local side should be in same subnet as remote peer's IP/Netmask combination. For example, you can set local IP to 192.168.1.1, and remote IP to 192.168.1.2, both having same netmask 255.255.255.0 .

Platforms

Windows
Mac OSX
Linux
BSD

LocalMAC property

Holds MAC address of local interface.

Type

String.

Syntax

C++

```
unsigned char LocalMAC[6];
```

Remarks

LocalMAC property holds MAC address for the adapter. MAC address is used by system sockets to correctly route raw network packets between network adapters (local adapters, and remote adapter). MAC address must be unique on the network. Changing this property does not really change MAC address of the adapter, it only changes how devP2P sees the adapter (unless you know what you're doing, do not change this value).

Local MAC value is sent to remote devP2P upon connection, since remote side needs our MAC to route packets to us.

Platforms

Windows
Mac OSX
Linux
BSD

LocalNetmask property

Holds netmask of local interface.

Type

Unsigned integer.

Syntax

C++

```
unsigned int LocalNetmask;
```

Remarks

LocalNetmask property holds subnet mask of the adapter. Typically it will be something like 255.255.255.0 . Both local devP2P peer and remote devP2P peer must have same netmask if you want them to exchange packets correctly. You should also make sure IP address for both sides is set to be inside same subnet. Changing this property value does not actually change adapter's netmask, except for devP2P. If you want to set IP/Netmask you should use windows tools, or right-click on the adapter and selecting Properties for it.

Platforms

Windows
Mac OSX
Linux
BSD

Name property

Returns name of the interface.

Type

String.

Syntax

C++

```
char Name[1024];
```

Remarks

Name property holds name of the virtual network adapter, as seen in Windows 'Network adapters' folder. You can change this value but that change will only affect devP2P, you cannot change real adapter's name.

Platforms

Windows
Mac OSX
Linux
BSD

SetIP method

Attempts to set local IP and netmask for the adapter.

Syntax

C++

```
void SetIP(char *ip, char *netmask);
```

The *SetIP(ip,netmask)* syntax has these parts:

ip	NULL terminated data buffer with new IP address.
netmask	NULL terminated data buffer with new IP address.

Remarks

SetIP will attempt to set new IP/Netmask values for the interface. It is assumed that process have root/Administrator privileges when calling this method. On Windwos, this method will call 'netsh' command, and on UNIXes it will call 'ifconfig' for setting up parameters.

Platforms

Windows
Mac OSX
Linux
BSD

Enumerations

ConnectionTypes	Lists type of connection established with remote peer.
Encryptions	List of encryption algorithms that can be used to protect the traffic.
Errors	List of errors that can be returned by devP2P.
ForwardTypes	Channel types available for port forwarding.
Protocols	Determines which protocol(s) can be used for connection between two devP2P instances.
States	List of possible devP2P states.

Platforms

Windows
Mac OSX
Linux
BSD

ConnectionTypes enumeration

Lists type of connection established with remote peer.

Remarks

Possible values for ConnectionTypes:

Constant	Value	Description
ConnectionUnspecified	0	Unknown connection type.
ConnectionTCPDirect	1	Direct TCP connection established with remote peer.
ConnectionUDPDirect	2	Direct UDP connection established with remote peer.
ConnectionTCPRelayed	3	Relayed TCP connection established with remote peer.

Platforms

Windows
Mac OSX
Linux
BSD

Encryptions enumeration

List of encryption algorithms that can be used to protect the traffic.

Remarks

Possible values for Encryptions:

Constant	Value	Description
EncNone	0	No encryption is used.
EncAES128	1	AES with 128bit key.
EncAES192	2	AES with 192bit key.
EncAES256	3	AES with 256bit key.

Platforms

Windows
Mac OSX
Linux
BSD

Errors enumeration

List of errors that can be returned by devP2P.

Remarks

Possible values for Errors:

Constant	Value	Description
ErrorNone	0	No error.
ErrorStopped	-1	devP2P is not started.
ErrorDisconnected	-2	Disconnected.
ErrorTimeout	-3	Timeout error.
ErrorNameNotSet	-4	Name not set.
ErrorCannotBind	-5	Cannot bind socket.
ErrorCancelled	-6	Cancelled.
ErrorAborted	-7	Aborted.
ErrorDisconnectedByRemote	-8	Disconnected by remote.
ErrorConnectionRefused	-9	Connection refused.
ErrorConnectionBroken	-10	Connection broken.
ErrorChannelInvalid	-11	Channel invalid.
ErrorChannelFull	-12	Channel full.
ErrorChannelBusy	-13	Channel busy.
ErrorCannotOpenFile	-14	Cannot open file.
ErrorCannotReadFile	-15	Cannot read from file.
ErrorCannotWritefile	-16	Cannot write to file.
ErrorForwardStarted	-17	Forward channel already started.
ErrorForwardInvalid	-18	Forward channel invalid.
ErrorCannotOpenAdapter	-19	Cannot open adapter.
ErrorLicenseInvalid	-20	License invalid.
ErrorNameInvalid	-21	Name invalid.
ErrorAdapterInvalid	-22	Adapter invalid.
ErrorAdapterInvalidIP	-23	Adapter has invalid IP.
ErrorAdapterExistingIP	-24	Adapter uses existing IP.

Platforms

Windows
Mac OSX
Linux
BSD

ForwardTypes enumeration

Channel types available for port forwarding.

Remarks

Possible values for ForwardTypes:

Constant	Value	Description
TCPLocalListen	0	Listens on local TCP port, forwards connection through devP2P to remote address.
TCPRemoteListen	1	Listens on remote TCP port, forwards connection through devP2P to local address.
UDPLocalListen	2	Listens on local UDP port, forwards connection through devP2P to remote address.
UDPRemoteListen	3	Listens on remote UDP port, forwards connection through devP2P to remote address.
TCPSocksProxy	4	Listens on local TCP port, forwards connection through devP2P to dynamicaly chosen address, using SOCKS4/4a/5 protocol.
TCPHttpProxy	5	Listens on local TCP port, forwards connection through devP2P to dynamicaly chosen address, WEB CONNECT protocol.

Platforms

Windows
Mac OSX
Linux
BSD

Protocols enumerations

Determines which protocol(s) can be used for connection between two devP2P instances.

Remarks

Possible values for Protocols:

Constant	Value	Description
ProtoTCP	1	Use TCP for transport protocol.
ProtoUDP	2	Use UDP for transport protocol.
ProtoBoth	3	Use both TCP and UDP for transport protocol.

Platforms

Windows
Mac OSX
Linux
BSD

States enumeration

List of possible devP2P states.

Remarks

Possible values for Protocols:

Constant	Value	Description
StateStopped	0	Stopped. Does not accept connections.
StateStarted	1	Started and idle listening. Accepts other devP2P connections.
StateSearching	2	Searching for remote devP2P peer.
StateSearchDone	3	Finished searching remote peer.
StateLinking	4	Attempting to link with remote devP2P peer.
StateConnected	5	Connected and linked with remote devP2P peer.
StateDisconnecting	6	Disconnecting from remote devP2P peer.
StateDisconnected	7	Disconnected from remote peer.

Platforms

Windows
Mac OSX
Linux
BSD