

SKSystemInfo

ActiveX Control

System Info ActiveX Control for Microsoft® Windows™

Copyright ©2001-2002 by Magneto Software, Inc.

All rights reserved

1 SkSystemInfo Overview	3
1.1 Introduction	3
1.2 Usage	3
1.3 Method Summary	4
2 Methods	5
2.1 AboutBox	5
2.2 GetAddressTranslationTableInfo	6
2.3 GetIcmpInfo	7
2.4 GetInterfacesTableInfo	8
2.5 GetIpAddressTableInfo	10
2.6 GetIpInfo	11
2.7 GetIpRoutingTableInfo	13
2.8 GetTcpConnectTableInfo	15
2.9 GetTcpInfo	16
2.10 GetUdpConnectTableInfo	17
2.11 GetUdpInfo	18

1 SkSystemInfo Overview

1.1 Introduction

SkSystemInfo ActiveX Control is a lightweight and powerful control that allows developers to retrieve some vital system information in real-time.

SkSystemInfo ActiveX Control provides the following real-time vital system information:

- Address translation (ARP) table information.
- ICMP protocol information and statistics.
- Networking interfaces table information and statistics.
- IP address table information.
- IP protocol information and statistics.
- IP routing table information.
- TCP connect table information.
- TCP protocol information and statistics.
- UDP connect table information.
- UDP protocol information and statistics.

It can be used from any 32-bit Windows development environment, including Visual Basic, Visual C++, and Delphi.

SkSystemInfo ActiveX Control is capable of processing multiple system information requests simultaneously.

It is fully compliant with RFC 1156.

SkSystemInfo ActiveX Control comes with all documentation, sample code, and working demo programs.

Additional information about data retrieved by SkSystemInfo ActiveX Control can be found at this location:

[RFC 1156 - Management Information Base for Network Management of TCP/IP-based Internets](#)

1.2 Usage

SkSystemInfo ActiveX Control can perform multiple requests simultaneously while providing information about critical networking information (ports usage, protocols statistics, and other information).

1.3 Method Summary

[AboutBox](#)

Display a dialog box with SkSystemInfo ActiveX Control license and version information.

[GetAddressTranslationTableInfo](#)

Retrieve the Address Translation group info.

[GetIcmpInfo](#)

Retrieve the ICMP group info.

[GetInterfacesTableInfo](#)

Retrieve the Interfaces table info.

[GetIpAddressTableInfo](#)

Retrieve the IP Address table info.

[GetIpInfo](#)

Retrieve the IP group info.

[GetIpRoutingTableInfo](#)

Retrieve the IP routing table info.

[GetTcpConnectTableInfo](#)

Retrieve the TCP connect table info.

[GetTcpInfo](#)

Retrieve the TCP group info.

[GetUdpConnectTableInfo](#)

Retrieve the UDP connect table info.

[GetUdpInfo](#)

Retrieve the UDP group info.

2 Methods

2.1 *AboutBox*

Summary

Display a dialog box with SkSystemInfo ActiveX Control license and version information.

Syntax

```
void AboutBox();
```

Description

This method could be used to display version license information or to register SKSYSTEMINFO.OCX control.

Parameters

None.

2.2 GetAddressTranslationTableInfo

Summary

Retrieve the Address Translation group info.

The Address Translation group contains one table, which is the union across all interfaces of the translation tables for converting a Network Address (e.g., an IP address) into a sub network-specific address.

Examples of such translation tables are: for broadcast media where ARP is in use, the translation table is equivalent to the ARP cache; or, on an X.25 network where non-algorithmic translation to X.121 addresses is required, the translation table contains the Network Address to X.121 address equivalences.

The Address Translation tables contain the NetworkAddress to "physical" address equivalences. Some interfaces do not use translation tables for determining address equivalences (e.g., DDN-X.25 has an algorithmic method); if all interfaces are of this type, then the Address Translation table is empty, i.e., has zero entries.

Syntax

long GetAddressTranslationTableInfo (VARIANT* *pvarAddressTranslationTableInfo*);

Description

The GetAddressTranslationTableInfo method retrieves the Address Translation group info.

It returns a long, which is set to 0 (ERROR_SUCCESS) if the method is successfully executed.

Parameters

pvarAddressTranslationTableInfo is the pointer to a variant, containing two-dimensional SAFEARRAY of data.

Each element of this SAFEARRAY is a VARIANT.

The SAFEARRAY column element indexes their definitions are as follows:

0 – atIfIndex, is the interface on which this entry's equivalence is effective. The interface identified by particular value of this index is the same interface as identified by the same value of ifIndex.

1 - atPhysAddress, the media-dependent "physical" addresses.

2 - atNetAddress, the NetworkAddress (e.g., the IP address) corresponding to the media-dependent "physical" address.

The number of rows is equal to the number of entries in the Address Translation table.

2.3 GetIcmpInfo

Summary

Retrieve the ICMP group info.

The ICMP group contains the ICMP input and output statistics.

Syntax

```
long GetIcmpInfo (VARIANT* pvarIcmpInfo);
```

Description

The GetIcmpInfo method retrieves the ICMP group info.

It returns a long, which is set to 0 (ERROR_SUCCESS) if the method is successfully executed.

Parameters

pvarIcmpInfo is the pointer to a variant, containing two-dimensional SAFEARRAY of data.

Each element of this SAFEARRAY is a VARIANT.

There are two columns in this array (column #0 – Description, column #1 – Value).

The SAFEARRAY row element indexes and their definitions are as follows:

- 0 – icmpInMsgs, the total number of ICMP messages, which the entity received. Note that this counter includes all those counted by icmpInErrors.
- 1 – icmpInErrors, the number of ICMP messages, which the entity received but determined as having errors (bad ICMP checksums, bad length, etc.).
- 2 – icmpInDestUnreachs, the number of ICMP Destination Unreachable messages received.
- 3 – icmpInTimeExcds, the number of ICMP Time Exceeded messages received.
- 4 – icmpInParmProbs, the number of ICMP Parameter Problem messages received.
- 5 – icmpInSrcQuenchs, the number of ICMP Source Quench messages received.
- 6 – icmpInRedirects, the number of ICMP Redirect messages received.
- 7 – icmpInEchos, the number of ICMP Echo (request) messages received.
- 8 – icmpInEchoReps, the number of ICMP Echo Reply messages received.
- 9 – icmpInTimestamps, the number of ICMP Timestamp (request) messages received.
- 10 – icmpInTimestampReps, the number of ICMP Timestamp Reply messages received.
- 11 – icmpInAddrMasks, the number of ICMP Address Mask Request messages received.
- 12 – icmpInAddrMaskReps, the number of ICMP Address Mask Reply messages received.
- 13 – icmpOutMsgs, the total number of ICMP messages which this entity attempted to send. Note that this counter includes all those counted by icmpOutErrors.
- 14 – icmpOutErrors, the number of ICMP messages, which this entity did not send due to problems discovered within ICMP such as a lack of buffers. This value should not include errors discovered outside the ICMP layer such as the inability of IP to route the resultant datagram.
- 15 – icmpOutDestUnreachs, the number of ICMP Destination Unreachable messages sent.
- 16 – icmpOutTimeExcds, the number of ICMP Time Exceeded messages sent.
- 17 – icmpOutParmProbs, the number of ICMP Parameter Problem messages sent.
- 18 – icmpOutSrcQuenchs, the number of ICMP Source Quench messages sent.
- 19 – icmpOutRedirects, the number of ICMP Redirect messages sent.
- 20 – icmpOutEchos, the number of ICMP Echo (request) messages sent.
- 21 – icmpOutEchoReps, the number of ICMP Echo Reply messages sent.
- 22 – icmpOutTimestamps, the number of ICMP Timestamp (request) messages sent.
- 23 – icmpOutTimestampReps, the number of ICMP Timestamp Reply messages sent.
- 24 – icmpOutAddrMasks, the number of ICMP Address Mask Request messages sent.
- 25 – icmpOutAddrMaskReps, the number of ICMP Address Mask Reply messages sent.

2.4 GetInterfacesTableInfo

Summary

Retrieve the Interfaces table info.

Syntax

```
long GetInterfacesTableInfo (VARIANT* pvarIfcTableInfo);
```

Description

The GetInterfacesTableInfo method retrieves the Interfaces table info.

It returns a long, which is set to 0 (ERROR_SUCCESS) if the method is successfully executed.

Parameters

pvarIfcTableInfo is the pointer to a variant, containing two-dimensional SAFEARRAY of data.

Each element of this SAFEARRAY is a VARIANT.

There are three columns in this array (column #0 – Interface index, column #1 – Parameter Description, column #2 – Parameter Value).

The SAFEARRAY row element indexes and their definitions are as follows:

0 – *ifIndex*, is a unique value for each interface. The value for each interface must remain constant at least from one re-initialization of the entity's network management system to the next re-initialization.

1 - *IfDescr*, a text string containing information about the interface. This string should include the name of the manufacturer, the product name and the version of the hardware interface. The string is intended for presentation to a human; it must not contain anything but printable ASCII characters.

2 – *ifType*, the type of interface, distinguished according to the physical/link/network protocol(s) immediately "below" IP in the protocol stack.

Syntax:

```
INTEGER {
    other(1),          -- none of the following
    regular1822(2),
    hdh1822(3),
    ddn-x25(4),
    rfc877-x25(5),
    ethernet-csmacd(6),
    iso88023-csmacd(7),
    iso88024-tokenBus(8),
    iso88025-tokenRing(9),
    iso88026-man(10),
    starLan(11),
    proteon-10MBit(12),
    proteon-80MBit(13),
    hyperchannel(14),
    fddi(15),
    lapb(16),
    sdlc(17),
    t1-carrier(18),
    cept(19),         -- European equivalent of T-1
    basicIsdn(20),
    primaryIsdn(21), -- proprietary serial
    propPointToPointSerial(22)
}
```

3 – *ifMtu*, the size of the largest IP datagram, which can be sent/received on the interface, specified in octets.

4 – *ifSpeed*, an estimate of the interface's current bandwidth in bits per second. For interfaces, which do not vary in bandwidth, or for those where no accurate estimation can be made, this object should contain the nominal bandwidth.

5 – *ifPhysAddress*, the interface's address at the protocol layer immediately "below" IP in the stack. For interfaces, which do not have such an address (e.g., a serial line), this object should contain an octet string of zero length.

6 - ifAdminStatus, the desired state of the interface. The testing (3) state indicates that no operational packets can be passed.

Syntax:

```
INTEGER {
    up(1),    -- ready to pass packets
    down(2),
    testing(3) -- in some test mode
}
```

7 – ifOperStatus, the current operational state of the interface. The testing (3) state indicates that no operational packets can be passed.

Syntax:

```
INTEGER {
    up(1),    -- ready to pass packets
    down(2),
    testing(3) -- in some test mode
}
```

8 – ifLastChange, the value of sysUpTime at the time the interface entered its current operational state. If the current state was entered prior to the last re-initialization of the local network management subsystem, then this object contains a zero value.

9 – ifInOctets, the total number of octets received on the interface, including framing characters.

10 – ifInUcastPkts, the number of (subnet) unicast packets delivered to a higher-layer protocol.

11 – ifInNUcastPkts, the number of non-unicast (i.e., subnet broadcast or subnet multicast) packets delivered to a higher-layer protocol.

12 – ifInDiscards, the number of inbound packets which were chosen to be discarded even though no errors had been detected to prevent their being deliverable to a higher-layer protocol. One possible reason for discarding such a packet could be to free up buffer space.

13 – ifInErrors, the number of inbound packets that contained errors preventing them from being deliverable to a higher-layer protocol.

14 – ifInUnknownProtos, the number of packets received via the interface, which were discarded because of an unknown or unsupported protocol.

15 – ifOutOctets, the total number of octets transmitted out of the interface, including framing characters.

16 – ifOutUcastPkts, the total number of packets that higher-level protocols requested be transmitted to a subnet-unicast address, including those that were discarded or not sent.

17 – ifOutNUcastPkts, the total number of packets that higher-level protocols requested be transmitted to a non-unicast (i.e., a subnet broadcast or subnet multicast) address, including those that were discarded or not sent.

18 – ifOutDiscards, the number of outbound packets, which were chosen to be discarded even though, no errors had been detected to prevent their being transmitted. One possible reason for discarding such a packet could be to free up buffer space.

19 – ifOutErrors, the number of outbound packets that could not be transmitted because of errors.

20 – ifOutQLen, the output packet queue length (in packets).

2.5 *GetIpAddressTableInfo*

Summary

Retrieve the IP Address Table info.

The IP Address table contains this entity's IP addressing Information

Syntax

```
long GetIpAddressTableInfo (VARIANT* pvarIpAddrTableInfo);
```

Description

The *GetIpAddressTableInfo* method retrieves the Address Translation group info.

It returns a long, which is set to 0 (ERROR_SUCCESS) if the method is successfully executed.

Parameters

pvarIpAddrTableInfo is the pointer to a variant, containing two-dimensional SAFEARRAY of data.

Each element of this SAFEARRAY is a VARIANT.

The SAFEARRAY column element indexes their definitions are as follows:

0 – *ipAdEntAddr*, is the IP address to which this entry's addressing information pertains.

1 – *ipAdEntIfIndex*, the index value that uniquely identifies the interface to which this entry is applicable.

The interface identified by a particular value of this index is the same interface as identified by the same value of *ifIndex*.

2 - *ipAdEntNetMask* he subnet mask associated with the IP address of this entry. The value of the mask is an IP address with all the network bits set to 1 and all the hosts' bits set to 0.

The number of rows is equal to the number of entries in the Address Translation table.

3 – *ipAdEntBcastAddr*, the value of the least-significant bit in the IP broadcast address used for sending datagrams on the (logical) interface associated with the IP address of this entry. For example, when the Internet standard all-ones broadcast address is used, the value will be 1.

2.6 GetIpInfo

Summary

Retrieve the IP group info.

Syntax

```
long GetIpInfo (VARIANT* pvarIpInfo);
```

Description

The GetIpInfo method retrieves the IP group info.

It returns a long, which is set to 0 (ERROR_SUCCESS) if the method is successfully executed.

Parameters

pvarIpInfo is the pointer to a variant, containing two-dimensional SAFEARRAY of data.

Each element of this SAFEARRAY is a VARIANT.

There are two columns in this array (column #0 – Description, column #1 – Value).

The SAFEARRAY row element indexes and their definitions are as follows:

0 – ipForwarding, the indication of whether this entity is acting as an IP gateway in respect to the forwarding of datagrams received by, but not addressed to, this entity. IP gateways forward datagrams; Hosts do not (except those Source-Routed via the host).

Syntax:

```
INTEGER {  
    gateway(1), -- entity forwards datagrams  
    host(2)    -- entity does NOT forward datagrams  
}
```

1 – ipDefaultTTL, the default value inserted into the Time-To-Live field of the IP header of datagrams originated at this entity, whenever a TTL value is not supplied by the transport layer protocol.

2 – ipInReceives, the total number of input datagrams received from interfaces, including those received in error.

3 – ipInHdrErrors, the number of input datagrams discarded due to errors in their IP headers, including bad checksums, version number mismatch, other format errors, time-to-live exceeded, errors discovered in processing their IP options, etc.

4 – ipInAddrErrors, the number of input datagrams discarded because the IP address in their IP header's destination field was not a valid address to be received at this entity. This count includes invalid addresses (e.g., 0.0.0.0) and addresses of unsupported Classes (e.g., Class E). For entities which are not IP Gateways and therefore do not forward datagrams, this counter includes datagrams discarded because the destination address was not a local address.

5 – ipForwDatagrams, the number of input datagrams for which this entity was not their final IP destination, as a result of which an attempt was made to find a route to forward them to that final destination. In entities which do not act as IP Gateways, this counter will include only those packets which were Source-Routed via this entity, and the Source-Route option processing was successful.

6 – IpInUnknownProtos, the number of locally addressed datagrams received successfully but discarded because of an unknown or unsupported protocol.

7 – ipInDiscards, the number of input IP datagrams for which no problems were encountered to prevent their continued processing, but which were discarded (e.g. for lack of buffer space). Note that this counter does not include any datagrams discarded while awaiting re-assembly.

8 – ipInDelivers, the total number of input datagrams successfully delivered to IP user-protocols (including ICMP).

9 – ipOutRequests, the total number of IP datagrams, which local IP user-protocols (including ICMP) supplied to IP in requests for transmission. Note that this counter does not include any datagrams counted in ipForwDatagrams.

10 – ipOutDiscards, the number of output IP datagrams for which no problem was encountered to prevent their transmission to their destination, but which were discarded (e.g., for lack of buffer space). Note that this counter would include datagrams counted in ipForwDatagrams if any such packets met this (discretionary) discard criterion.

11 – ipOutNoRoutes, the number of IP datagrams discarded because no route could be found to transmit them to their destination. Note that this counter includes any packets counted in ipForwDatagrams, which meet this "no-route" criterion.

12 – ipReasmTimeout, the maximum number of seconds, which received fragments, are held while they are awaiting reassembly at this entity.

13 – ipReasmReqds, the number of IP fragments received which needed to be reassembled at this entity.

14 – ipReasmOKs, the number of IP datagrams successfully re-assembled.

15 – ipReasmFails, the number of failures detected by the IP re-assembly algorithm (for whatever reason: timed out, errors, etc). Note that this is not necessarily a count of discarded IP fragments since some algorithms (notably RFC 815's) can lose track of the number of fragments by combining them as they are received.

16 – ipFragOKs, the number of IP datagrams that have been successfully fragmented at this entity.

17 – ipFragFails, the number of IP datagrams that have been discarded because they needed to be fragmented at this entity but could not be, e.g., because their "Don't Fragment" flag was set.

18 – ipFragCreates, the number of IP datagram fragments that have been generated as a result of fragmentation at this entity.

2.7 GetIpRoutingTableInfo

Summary

Retrieve the IP routing table info.

The IP Routing Table contains an entry for each route presently known to this entity. Note that the action to be taken in response to a request to read a non-existent entry is specific to the network management protocol being used.

Syntax

```
long GetIpRoutingTableInfo (VARIANT* pvarIpRoutingTableInfo);
```

Description

The GetIpRoutingTableInfo method retrieves the Address Translation group info.

It returns a long, which is set to 0 (ERROR_SUCCESS) if the method is successfully executed.

Parameters

pvarIpRoutingTableInfo is the pointer to a variant, containing two-dimensional SAFEARRAY of data.

Each element of this SAFEARRAY is a VARIANT.

The SAFEARRAY column element indexes their definitions are as follows:

0 – ipRouteDest, is the destination IP address of this route. An entry with a value of 0.0.0.0 is considered a default route. Multiple such default routes can appear in the table, but access to such multiple entries is dependent on the table-access mechanisms defined by the network management protocol in use.

1 – ipRouteIfIndex, the index value, which uniquely identifies the local interface through which the next hop of this route, should be reached. The interface identified by a particular value of this index is the same interface as identified by the same value of ifIndex.

2 - ipRouteMetric1, the primary routing metric for this route. The semantics of this metric are determined by the routing-protocol specified in the route's ipRouteProto value. If this metric is not used, its value will be set to -1.

3 - ipRouteMetric2, an alternate routing metric for this route. The semantics of this metric are determined by the routing-protocol specified in the route's ipRouteProto value. If this metric is not used, its value will be set to -1.

4 - ipRouteMetric3, an alternate routing metric for this route. The semantics of this metric are determined by the routing-protocol specified in the route's ipRouteProto value. If this metric is not used, its value will be set to -1.

5 - ipRouteMetric4, an alternate routing metric for this route. The semantics of this metric are determined by the routing-protocol specified in the route's ipRouteProto value. If this metric is not used, its value will be set to -1.

6-ipRouteNextHop, is the IP address of the next hop of this route.

7 – ipRouteType, the type of route.

Syntax:

```
INTEGER {
    other(1),    -- none of the following
    invalid(2), -- an invalidated route
                -- route to directly
    direct(3),   -- connected (sub-)network
                -- route to a non-local
    remote(4),  -- host/network/sub-network
}
```

8 – ipRouteProto, the routing mechanism via which this route was learned. Inclusion of values for gateway routing protocols is not intended to imply that hosts should support those protocols.

Syntax:

```
INTEGER {
    other(1),    -- none of the following
                -- non-protocol information,
                -- e.g., manually configured
    local(2),   -- entries
                -- set via a network management
```

```
netmngmt(3), -- protocol
              -- obtained via ICMP,
icmp(4),     -- e.g., Redirect
              -- the remaining values are
              -- all gateway routing protocols
egp(5),
ggp(6),
hello(7),
rip(8),
is-is(9),
es-is(10),
ciscoIgrp(11),
bbnSpfIgp(12),
oigp(13)
}
```

9 – ipRouteAge, the number of seconds since this route was last updated or otherwise determined to be correct. Note that no semantics of "too old" can be implied except through knowledge of the routing protocol by which the route was learned.

The number of rows is equal to the number of entries in the IP routing table.

2.8 GetTcpConnectTableInfo

Summary

Retrieve the TCP connect table info. TCP connect table is a table containing TCP connection-specific information.

Syntax

```
long GetTcpConnectTableInfo (VARIANT* pvarTcpConnTableInfo);
```

Description

The GetTcpConnectTableInfo method retrieves the TCP connect table info.

It returns a long, which is set to 0 (ERROR_SUCCESS) if the method is successfully executed.

Parameters

pvarTcpConnTableInfo is the pointer to a variant, containing two-dimensional SAFEARRAY of data.

Each element of this SAFEARRAY is a VARIANT.

Each row represents a particular current TCP connection.

TCP connection is transient, in that it ceases to exist when (or soon after) the connection makes the transition to the CLOSED state.

The SAFEARRAY column element indexes their definitions are as follows:

0 – tcpConnProtName, the protocol name (“TCP”)

1 – tcpConnLocalAddress, the local IP address for this TCP connection.

2 – tcpConnLocalPort, the local port number for this TCP connection.

3 – tcpConnRemAddress, the remote IP address for this TCP connection.

4 – tcpConnRemPort, the remote port number for this TCP connection.

5 – tcpConnState, is the state of this TCP connection.

Syntax:

```
INTEGER {  
    closed(1),  
    listen(2),  
    synSent(3),  
    synReceived(4),  
    established(5),  
    finWait1(6),  
    finWait2(7),  
    closeWait(8),  
    lastAck(9),  
    closing(10),  
    timeWait(11)  
}
```

6 – tcpConnStateDescr, user-friendly description of the tcpConnState parameter.

The number of rows is equal to the number of entries in the TCP connect table.

2.9 GetTcpInfo

Summary

Retrieve the TCP group info.

Syntax

```
long GetTcpInfo (VARIANT* pvarTcpInfo);
```

Description

The GetTcpInfo method retrieves the TCP group info.

It returns a long, which is set to 0 (ERROR_SUCCESS) if the method is successfully executed.

Parameters

pvaTcpInfo is the pointer to a variant, containing two-dimensional SAFEARRAY of data.

Each element of this SAFEARRAY is a VARIANT.

There are two columns in this array (column #0 – Description, column #1 – Value).

The SAFEARRAY row element indexes and their definitions are as follows:

0 – tcpRtoAlgorithm, the algorithm used to determine the timeout value used for retransmitting unacknowledged octets.

Syntax:

```
INTEGER {  
    other(1), -- none of the following  
    constant(2), -- a constant rto  
    rsre(3), -- MIL-STD-1778, Appendix B  
    vanj(4) -- Van Jacobson's algorithm [15]  
}
```

1 – tcpRtoMin, the minimum value permitted by a TCP implementation for the retransmission timeout, measured in milliseconds. More refined semantics for objects of this type depend upon the algorithm used to determine the retransmission timeout. In particular, when the timeout algorithm is rsre(3), an object of this type has the semantics of the LBOUND quantity described in RFC 793.

2 – TcpRtoMax, the maximum value permitted by a TCP implementation for the retransmission timeout, measured in milliseconds. More refined semantics for objects of this type depend upon the algorithm used to determine the retransmission timeout. In particular, when the timeout algorithm is rsre(3), an object of this type has the semantics of the UBOUND quantity described in RFC 793.

3 – tcpMaxConn, the limit on the total number of TCP connections the entity can support. In entities where the maximum number of connections is dynamic, this object should contain the value "-1".

4 – tcpActiveOpens, the number of times TCP connections have made a direct transition to the SYN-SENT state from the CLOSED state.

5 – tcpPassiveOpens, the number of times TCP connections have made a direct transition to the SYN-RCVD state from the LISTEN state.

6 – tcpAttemptFails, the number of times TCP connections have made a direct transition to the CLOSED state from either the SYN-SENT state or the SYN-RCVD state, plus the number of times TCP connections have made a direct transition to the LISTEN state from the SYN-RCVD state.

7 – tcpEstabResets, the number of times TCP connections have made a direct transition to the CLOSED state from either the ESTABLISHED state or the CLOSE-WAIT state.

8 – tcpCurrEstab, the number of TCP connections for which the current state is either ESTABLISHED or CLOSE-WAIT.

9 – tcpInSegs, the total number of segments received, including those received in error. This count includes segments received on currently established connections.

10 – tcpOutSegs, the total number of segments sent, including those on current connections but excluding those containing only retransmitted octets. TcpRetransSegs, the total number of segments retransmitted - that is, the number of TCP segments transmitted containing one or more previously transmitted octets.

2.10 GetUdpConnectTableInfo

Summary

Retrieve the UDP connect table info. UDP connect table is a table containing UDP connection-specific information.

Syntax

```
long GetUdpConnectTableInfo (VARIANT* pvarUdpConnTableInfo);
```

Description

The `GetUdpConnectTableInfo` method retrieves the UDP connect table info.

It returns a long, which is set to 0 (ERROR_SUCCESS) if the method is successfully executed.

Parameters

pvarUdpConnTableInfo is the pointer to a variant, containing two-dimensional SAFEARRAY of data.

Each element of this SAFEARRAY is a VARIANT.

Each row represents a particular opened UDP port.

The SAFEARRAY column element indexes their definitions are as follows:

0 – `tcpConnProtName`, the protocol name (“UDP”)

1 – `udpLocalAddress`, the local IP address for this UDP connection.

2 – `udpLocalPort`, the local port number for this UDP connection.

The number of rows is equal to the number of entries in the UDP connect table.

2.11 *GetUdpInfo*

Summary

Retrieve the UDP group info.

Syntax

```
long GetUdpInfo (VARIANT* pvarUdpInfo);
```

Description

The *GetUdpInfo* method retrieves the UDP group info.

It returns a long, which is set to 0 (ERROR_SUCCESS) if the method is successfully executed.

Parameters

pvaUdpInfo is the pointer to a variant, containing two-dimensional SAFEARRAY of data.

Each element of this SAFEARRAY is a VARIANT.

There are two columns in this array (column #0 – Description, column #1 – Value).

The SAFEARRAY row element indexes and their definitions are as follows:

0 – *udpInDatagrams*, the total number of UDP datagrams delivered to UDP users.

1 – *udpNoPorts*, the total number of received UDP datagrams for which there was no application at the destination port.

2 – *udpInErrors*, the number of received UDP datagrams that could not be delivered for reasons other than the lack of an application at the destination port.

3 – *udpOutDatagrams*, the total number of UDP datagrams sent from this entity.